Проектування інформаційних систем

Марченко Анна Вікторівна

2016

Зміст

Проектування інформаційних систем

Ключові терміни:

*, С15

1.1 КЛАСИФІКАЦІЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Інформаційні системи можуть значно різнитися за типами об’єктів управління, характером та обсягом розв’язуваних задач і рядом інших ознак, тому їх можна класифікувати за такими ознаками.

1. За рівнем або сферою діяльності – державні, територіальні (регіональні), галузеві, об’єднань, підприємств або установ, технологічних процесів.

Державні ІС призначені для складання перспективних та поточних планів розвитку країни, обліку результатів та регулювання діяльності окремих ланцюгів народного господарства, розроблюють державний бюджет та контролюють його виконання і т.ін. До них відносяться автоматизована система державної статистики (АСДС), автоматизована система планових розрахунків (АСПР). державна інформаційна система фінансових розрахунків (АСФР) при Міністерстві фінансів України, система обробки інформації з цін (АСОІ цін), система управління національним банком (АСУ банк), система обробки науково-технічної інформації (АСО НТІ) і т.ін.

Територіальні (регіональні) ІС призначені для управління адміністративно-територіальним регіоном. Сюди належать ІС області, міста, району. Ці системи виконують роботи з обробки інформації, яка необхідна для реалізації функцій управління регіоном, формування звітності й видачі оперативних даних місцевим і керівним державним та господарським органам.

Галузеві інформаційні системи управління призначені для управління підвідомчими підприємствами та організаціями. Галузеві ІС діють у промисловості та в сільському господарстві, будівництві на транспорті і т.ін. В них розв’язуються задачі інформаційного обслуговування апарату управління галузевих міністерств і їх підрозділів. Галузеві ІС відрізняються сферами застосування – промислова, непромислова, наукова.

Інформаційні системи управління підприємствами (ІСУП) або виробничими об’єднаннями (ІСУ ВО) – це системи із застосуванням сучасних засобів автоматизованої обробки даних, економіко-математичних та інших методів для регулярного розв’язування задач управління виробничо-господарського діяльністю підприємства.

Інформаційні системи управління технологічними процесами (ІСУ ТП) керують станом технологічних процесів (робота верстата, домни тощо). Перша й головна відмінність цих систем від розглянутих раніше полягає передусім у характері об’єкта управління – для ІСУ ТП це різноманітні машини, прилади, обладнання, а для державних, територіальних та інших АСУ – це колективи людей. Друга відмінність полягає у формі передачі інформації. Для ІСУ ТП основною формою передачі інформації є сигнал, а в інших ІСУ – документи.

2. За рівнем автоматизації процесів управління – інформаційно-пошукові, інформаційно-довідкові, інформаційно-керівні, системи підтримки прийняття рішень, інтелектуальні ІС.

Інформаційно-пошукові системи (ІСП) орієнтовані на розв’язування задач пошуку інформації. Змістова обробка інформації у таких системах відсутня.

В інформаційно-довідкових системах (ІДС) за результатами пошуку обчислюють значення арифметичних функцій.

Інформаційно-управляючі, або управлінські, системи (відомі у вітчизняній літературі під назвою «автоматизовані системи організаційного управління») являють собою організаційно-технічні системи, які забезпечують вироблення рішення на основі автоматизації інформаційних процесів у сфері управління. Отже, ці системи призначені для автоматизованого розв’язування широкого кола задач управління.

До інформаційних систем нового покоління належать системи підтримки прийняття рішень (СППР) та інформаційні системи, побудовані на штучному інтелекті (інтелектуальні ІС).

СППР – це інтерактивна комп’ютерна система, яка призначена для підтримки різних видів діяльності при прийнятті рішень із слабоструктурованих або неструктурованих проблем.

Інтерес до СППР, як перспективної галузі використання обчислювальної техніки та інструментарію підвищення ефективності праці у сфері управління економікою, постійно зростає. У багатьох країнах розробка та реалізація СППР перетворилася на дільницю бізнесу, що швидко

розвивається.

Штучний інтелект – це штучні системи, створені людиною на базі ЕОМ, що імітують розв'язування людиною складних творчих задач. С15 Створенню інтелектуальних інформаційних систем сприяла розробка в теорії штучного інтелекту логіко-лінгвістичних моделей. Ці моделі дають змогу формалізувати конкретні змістовні знання про об'єкти управління та процеси, що відбуваються в них, тобто ввести в ЕОМ логіко-лінгвістичні моделі поряд з математичними. Логіко-лінгвістичні моделі – це семантичні мережі, фрейми, продукційні системи – іноді об'єднуються терміном «програмно-апаратні засоби в системах штучного інтелекту».

Розрізняють три види інтелектуальних ІС:

- інтелектуальні інформаційно-пошукові системи (системи типу «запитання – відповідь»), які у процесі діалогу забезпечують взаємодію кінцевих користувачів-непрограмістів з базами даних та знань професійними мовами користувачів, близьких до природних;
- розрахунково-логічні системи, які дають змогу кінцевим користувачам, що не є програмістами та спеціалістами в галузі прикладної математики, розв'язувати в режимі діалогу з ЕОМ свої задачі з використанням складних методів і відповідних прикладних програм;
- експертні системи, які дають змогу провадити ефективну комп'ютеризацію областей, в яких знання можуть бути подані в експертній описовій формі, але використання математичних моделей утруднене або неможливе.

В економіці України найпоширенішими є експертні системи. Це системи, які дають змогу на базі сучасних персональних комп'ютерів виявляти, нагромаджувати та коригувати знання з різних галузей народного господарства (предметних областей).

3. За ступенем централізації обробки інформації – централізовані ІС, децентралізовані ІС, інформаційні системи колективного використання.

4. За ступенем інтеграції функцій – багаторівневі ІС з інтеграцією за рівнями управління (підприємство – об'єднання, об'єднання – галузь і т.ін.), багаторівневі ІС з інтеграцією за рівнями планування і т.ін.

5. За типом ІС розподіляються на фактографічні, документальні і документально-фактографічні ІС.

Документальна ІС – це система, в якій об'єктом зберігання і обробки є власне документи.

Фактографічна ІС – це система, в якій, об'єктом або сутністю є дещо, що являє для проблемної сфери багатосторонній інтерес (співробіник, договір, виріб тощо). Відомості про ці сутності можуть знаходитись у множині різних вхідних і вихідних повідомлень.

1.2 МЕТА, ЗАДАЧІ ТА ПРИНЦИПИ СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Мета створення інформаційних систем – у гранично короткі терміни створити систему обробки даних, яка має задані споживчі якості. До них належать: функціональна повнота, своєчасність, функціональна надійність, адаптивна надійність, економічна ефективність.

Функціональна повнота – це властивість інформаційної системи, яка характеризує рівень автоматизації управлінських робіт.

Коефіцієнт функціональної повноти

$$K_f = \frac{П_a}{П_o},$$

де $П_a$ – показники, отримувані автоматизовано; $П_o$ – загальна кількість показників.

Своєчасність – це властивість інформаційної системи, яка характеризує можливість отримання апаратом керівництва необхідної інформації.

Коефіцієнт своєчасності

$$K_c = \frac{П_a - П_a}{П_a},$$

де $П_a$ – кількість показників, отриманих із затримкою щодо планового терміну подання.

Функціональна надійність – це властивість інформаційної системи виконувати свої функції з обробки даних. * Це сукупність надійностей програмного, інформаційного та технічного забезпечення.

Адаптивна надійність – це властивість інформаційної системи виконувати свої функції, якщо вони змінюються в межах умов, зумовлених розвитком системи керування об'єкта впродовж заданого проміжку часу.

Економічна ефективність інформаційної системи виявляється в покращенні економічних результатів функціонування об'єкта в результаті впровадження інформаційної системи.

Створення інформаційної системи передбачає частковий чи повний перегляд методів і засобів функціонування інформаційної системи економічного об'єкта і виконання таких завдань.

1. Виявлення його суттєвих характеристик.
2. Створення математичних і фізичних моделей досліджуваної системи та її елементів.
3. Встановлення умов взаємодії людини та комплексу технічних засобів.
4. Детальна розробка окремих проектних рішень.
5. Аналіз проектних рішень, практична апробація та впровадження.

Перше що потрібно зробити це вивчити питання доцільності створення інформаційної системи, що проходить декілька етапів показаних на рис. 1.1.

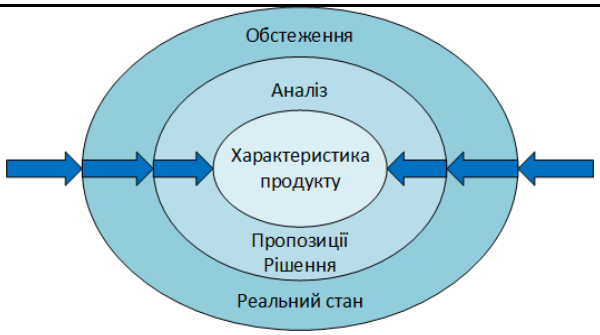


Рисунок 1.1 – Прийняття рішення про доцільність створення ІС

Інформаційну систему створюють у тих випадках, коли потрібно організувати нові обчислювальні центри, вдосконалити діючу методику й техніку розв'язання задач, впровадити нові задачі, а також організувати інформаційну систему.

Принципи створення інформаційної системи поділяють на дві частини: загальні та часткові.

Загальні принципи мають універсальний характер і визначають методологічний підхід до створення будь-яких об'єктів. Це такі принципи: науковості, нормативності, неперервності, розвитку, ефективності, послідовності, від загального до часткового, системний, комплексності, використання типових і керівних матеріалів.

Часткові принципи: систему управління потрібно розглядати як людино-машинну; чіткий поділ системи на складові, забезпечення сумісності й зв'язку між усіма видами забезпечення; забезпечення єдності обліку, типізація, уніфікація та стандартизація.

При створенні інформаційної системи треба керуватися принципами, визначеними *РД 50–680–88 «АС Основные положения»*: системності, розвитку (відкритості), сумісності, стандартизації (уніфікації) та ефективності.

Принцип системності: при декомпозиції мають бути встановлені такі зв'язки між структурними елементами системи, які забезпечують цілісність інформаційної системи та її взаємодію з іншими системами.

Принцип розвитку (відкритості): виходячи із перспектив розвитку об'єкта автоматизації інформаційну систему треба створювати з урахуванням можливості поповнення та обновлення функцій і складу інформаційної системи, не порушуючи її функціонування.

Принцип сумісності: при створенні систем мають бути реалізовані інформаційні інтерфейси, завдяки яким вона може взаємодіяти з іншими системами за встановленими правилами.

Принцип стандартизації (уніфікації): при створенні систем мають бути раціонально використані типові, уніфіковані й стандартизовані елементи, проектні рішення, пакети прикладних програм, комплекси, компоненти.

Принцип ефективності: досягнення раціонального співвідношення між затратами і цільовими ефектами, включаючи кінцеві результати, отримані завдяки автоматизації.

Однією з основних умов створення високоефективної інформаційної системи є орієнтація на користувача. При функціонуванні інформаційної системи, розв'язанні завдань управління діє велика кількість обмежень, які потрібно враховувати під час її розробки. Крім того, в процесі самого проектування виникає багато обмежень. Це призводить до того, що в пошуках найкращого шляху, за який часто беруть найбільш простий, швидкий і дешевий, розробники свідомо чи підсвідомо перекладають частину проблем, що виникли, на користувача. Цей шлях може призвести до згубних наслідків. Користувачі, в свою чергу, прагнучи мінімізувати обсяги своєї роботи, не виконують інструкцій розробника й ігнорують систему, яка не полегшує, а ускладнює їм життя. При цьому слід ураховувати основну особливість об'єкта: до створення інформаційної системи завдання управління можуть розв'язуватись «вручну», без використання ЕОМ. Тому основне питання в якості та ефективності рішень, які приймаються. Отож інколи інформаційна система функціонує сама по собі, а управління об'єктом здійснюється майже без неї. Інформаційна система має бути інструментом управління, в якому основну роль відіграє людина. Сам процес повинен не зводитись до створення інформаційної системи, як самостійного продукту, але і забезпечити його документацією, гарантією і супроводженням рис. 1.2.

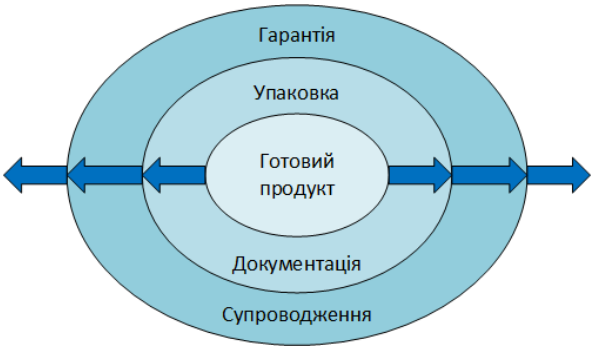


Рисунок 1.2 – Створення ІС

1.3 ПРОЦЕСИ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Поняття життєвого циклу програмного забезпечення (ЖЦ ПЗ) є одним з базових у програмній інженерії (ПІ).

Життєвий цикл ПЗ – певна послідовність фаз або стадій від моменту прийняття рішення про необхідність створення ПЗ до повного вилучення ПЗ з експлуатації.

На кожній фазі відбувається певна сукупність процесів, кожний з яких породжує певний продукт, використовуючи необхідні ресурси. Стандарт міжнародної організації ISO/IEC 12207:1995 "Information Technology – Software Life Cycle Processes" визначає структуру ЖЦ, що містить процеси, дії і задачі, які мають бути виконані під час створення ПЗ.

Стандарт визначає програмне забезпечення як набір комп'ютерних програм, процедур і, можливо, пов'язаних із ними документації й даних. Процес – це сукупність взаємопов'язаних дій, що перетворюють вхідні дані у вихідні.

Процес поділяється на набір дій, а дії – на набір задач. Процеси, дії та задачі ініціюються іншими процесами і виконуються у міру необхідності, причому немає заздалегідь визначених послідовностей виконання.

Усі продукти програмної інженерії становлять певні описи – тексти вимог до розроблення, узгодження домовленостей, документацію, тексти програм, інструкції щодо експлуатації тощо. Головні ресурси програмної інженерії, що визначають ефективність розроблень, – це час та вартість.

Відповідно до стандарту ISO/IEC 12207 усі процеси ЖЦ ПЗ поділяються на три групи (рис. 1.3):

- основні процеси (придбання, доставка, розроблення, експлуатація, супровід);
- організаційні процеси (управління, удосконалення, навчання);
- допоміжні процеси (документування, забезпечення якості, верифікація, атестація, аудит, загальна оцінка тощо).

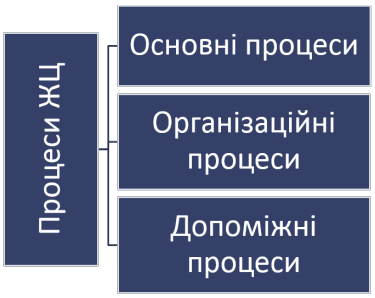


Рисунок 1.3 – Процеси життєвого циклу розроблення ПЗ

Основні процеси включають:

- процес придбання, що ініціює життєвий цикл ІС та визначає її покупця; передбачають виконання замовлення та постачання продукту замовнику.
- процес розроблення, що визначає дії організації – розробника інформаційного продукту; передбачає дії, що виконуються розробником, і охоплює роботи зі створення ПЗ та його компонентів відповідно до вимог, включаючи оформлення проектної й експлуатаційної документації, підготовку матеріалів, необхідних для перевірки працездатності і відповідної якості програмних продуктів, матеріалів, потрібних для організації навчання персоналу.
- процес постачання, що визначає дії під час передачі розробленого продукту покупцеві;
- процес експлуатації, що означає дії з обслуговування системи під час її використання – консультації користувачів, вивчення їхніх побажань тощо;
- процес супроводження, що означає дії з керування модифікаціями, підтримки актуального стану та функціональної придатності, інсталяції та вилучення версій систем у користувача.

Процес розроблення ПЗ має забезпечити шлях від усвідомлення потреб замовника до передачі йому готового продукту (рис. 1.4). Він складається з таких етапів:

- визначення вимог – збір та аналіз вимог замовника виконавцем та подання їх у нотації, що зрозуміла як замовнику, так і виконавцю;

Рисунок 1.4 – Процеси розроблення програмного забезпечення

- проектування – перетворення вимог до розроблення у послідовність проектних рішень щодо способів реалізації вимог: формування загальної архітектури програмної системи та принципів її прив'язки до конкретного середовища функціонування; визначення детального складу модулів кожної з архітектурних компонент;
- реалізація – перетворення проектних рішень у програмну систему, що реалізує означені рішення;
- тестування – перевірка кожного з модулів та способів їх інтеграції; тестування програмного продукту в цілому (так звана верифікація); тестування відповідності функцій працюючої програмної системи вимогам, що були до неї поставлені замовником (так звана валідація);

- експлуатація та супроводження готової системи.

Підготовча робота починається з вибору моделі ЖЦ ПЗ, що відповідає масштабові, значимості і складності проекту. Процес розроблення має відповідати обраній моделі. Розробник повинен вибрати, адаптувати до умов проекту і використовувати погоджені із замовником стандарти, методи й засоби розроблення, а також скласти план виконання робіт.

Аналіз вимог до системи розглядає функціональні можливості, вимоги користувача, вимоги до надійності і безпеки, вимоги до зовнішніх інтерфейсів тощо. Вимоги до системи оцінюються відповідно до критеріїв реалізації і можливості перевірки при тестуванні.

Проектування архітектури системи полягає у визначенні компонентів її устаткування, ПЗ й операцій, що виконуються персоналом.

Аналіз вимог до ПЗ визначає: функціональні можливості, включаючи характеристики продуктивності і середовища функціонування компонента; зовнішні інтерфейси; специфікації надійності і безпеки; ергономічні вимоги; вимоги до даних; вимоги до інсталяції та введення системи; вимоги до документації користувачів; вимоги до експлуатації і супроводу.

Проектування архітектури ПЗ включає такі задачі (для кожного компонента ПЗ):

- трансформацію вимог до ПЗ в архітектуру, що визначає структуру ПЗ і склад його компонентів;
- розроблення і документування програмних інтерфейсів ПЗ і БД;
- розроблення попередньої версії документації користувачів;
- розроблення і документування попередніх вимог до тестів і плану інтеграції ПЗ.

Детальне проектування ПЗ включає такі задачі:

- опис компонентів ПЗ й інтерфейсів між ними на нижчому рівні, що достатній для їх подальшого самостійного кодування і тестування;
- розроблення і документування детального проекту бази даних;
- відновлення (за необхідності) документації;
- розроблення і документування вимог до тестів і плану тестування компонентів ПЗ;
- відновлення плану інтеграції ПЗ.

Кодування і тестування ПЗ охоплюють такі задачі:

- розроблення (кодування) і документування кожного компонента ПЗ і бази даних, а також сукупності тестових процедур і даних для їхнього тестування
- тестування кожного компонента ПЗ і БД на відповідність вимогам. Результати тестування компонентів мають бути документовані
- відновлення (за необхідності) документації користувачів
- відновлення плану інтеграції ПЗ

Інтеграція ПЗ передбачає збирання розроблених компонентів ПЗ відповідно до плану інтеграції і тестування компонентів. Для кожного з компонентів розробляються набори тестів і тестові процедури, що призначені для перевірки кваліфікаційних вимог при наступному кваліфікаційному тестуванні. Кваліфікаційна вимога – це набір критеріїв або умов, який необхідно виконати, щоб кваліфікувати програмний продукт на відповідність своїм специфікаціям і можливість його використовувати в умовах експлуатації.

Кваліфікаційне тестування ПЗ проводиться розробником у присутності замовника для демонстрації того, що ПЗ дійсно відповідає своїм специфікаціям. Кваліфікаційне тестування здійснюється для кожного компонента ПЗ щодо всіх вимог при використанні різних тестів. При цьому також перевіряються повнота технічної документації та її адекватність самим компонентам ПЗ.

Інтеграція системи полягає в об'єднанні всіх її компонентів, включно з ПЗ й устаткуванням. Після інтеграції система у свою чергу піддається кваліфікаційному тестуванню на відповідність сукупності вимог до неї. При цьому також готуються оформлення і перевірка повного комплекту документації на систему.

Встановлення ПЗ здійснюється розробником відповідно до плану в тому операційному середовищі і на тому обладнанні, що передбачені замовленням.

Приймання ПЗ передбачає оцінку результатів кваліфікаційного тестування ПЗ та системи і документування результатів оцінювання, що проводиться замовником за допомогою розробника. Розробник здійснює остаточну передачу ПЗ замовнику відповідно до договору, забезпечуючи при цьому необхідне навчання і підтримку.

Процес експлуатації охоплює дії і задачі оператора-організації, що експлуатує систему. Цей процес включає такі етапи: 1) підготовчу роботу; 2) експлуатаційне тестування; 3) експлуатацію системи; 4) підтримку користувачів.

Підготовча робота включає проведення оператором планування дій і робіт, що виконуються у процесі експлуатації, й установку експлуатаційних стандартів та визначення процедур локалізації і розв'язання проблем, які виникають у процесі експлуатації.

Експлуатаційне тестування проводиться для кожної чергової версії програмного продукту, після чого вона передається в експлуатацію.

Експлуатація системи здійснюється у призначеній для цього ОС відповідно до документації користувачів.

Підтримка користувачів полягає в наданні допомоги і консультацій при виявленні помилок у процесі експлуатації ПЗ.

Процес супроводу передбачає дії і задачі, що виконуються службою супроводу. Цей процес активізується при модифікаціях програмного продукту і відповідної документації або модернізації, адаптації ПЗ. Супровід – це внесення змін у ПЗ з метою виправлення помилок, підвищення продуктивності або адаптації до умов праці, що змінилися.

Зміни, внесені в наявне ПЗ, не повинні порушувати його цілісність. Процес супроводу включає перенесення ПЗ в інше середовище (міграцію) і закінчується зняттям ПЗ з експлуатації. Цей процес охоплює такі дії: 1) підготовчу роботу; 2) аналіз проблем і запитів на модифікацію ПЗ; 3) модифікацію ПЗ; 4) перевірку і приймання; 5) міграцію ПЗ в інше середовище; 6) зняття ПЗ з експлуатації.

Підготовча робота служби супроводу включає такі задачі: планування дій і робіт, які виконуються у процесі супроводу та визначення процедур локалізації і розв'язання проблем, що виникають у процесі супроводу.

Аналіз проблем і запитів на модифікацію ПЗ що виконуються службою супроводу, включає такі задачі:

- аналіз повідомлення про проблему або запит на модифікацію ПЗ. При цьому визначаються такі характеристики можливої модифікації: тип (коригувальна, поліпшуюча, профілактична); масштаб (розміри модифікації, вартість і термін її реалізації); критичність (вплив на продуктивність, надійність або безпеку);
- оцінка доцільності та можливих варіантів проведення модифікації;
- затвердження обраного варіанта модифікації.

Модифікація ПЗ передбачає визначення компонентів ПЗ, їхніх версій і документації, що підлягають модифікації, внесення необхідних змін відповідно до правил процесу розроблення. Підготовлені зміни тестуються і перевіряються за критеріями, що передбачені документацією. При підтвердженні коректності змін у програмах відбувається коригування документації.

Перевірка і приймання полягають у перевірці цілісності модифікованої системи і затвердженні внесених змін.

При перенесенні ПЗ в інше середовище використовуються наявні або розробляються нові засоби перенесення, потім виконується конвертування програм і даних у нове середовище. З метою полегшення переходу передбачається паралельна експлуатація ПЗ у старому і новому середовищі впродовж певного періоду, під час якого проводиться необхідне навчання користувачів з новим ПЗ.

Зняття ПЗ з експлуатації здійснюється за рішенням замовника за участю організації експлуатації, служби супроводу і користувачів. При цьому програмні продукти і відповідна документація підлягають архівуванню відповідно до договору.

1.4. МОДЕЛІ ЖИТТЄВОГО ЦИКЛУ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.4.1. Поняття і моделі життєвого циклу

У програмній інженерії термін «життєвий цикл» застосовується до штучних систем ПЗ і означає зміни, які відбуваються в «житті» програмного продукту.

Різні стадії між «народженням» виробу і його можливої «смертю» відомі як стадії життєвого циклу.

Найбільш часто говорять про такі моделі життєвого циклу:

- каскадна (водоспадна) або послідовна;
- ітеративна і інкрементально-еволюційна (гібридна, змішана);
- спіральна (модель Боема).

Легко виявити, що в різний час і в різних джерелах наводиться різний список моделей та їх інтерпретація. Наприклад, раніше, інкрементальна модель розумілася як побудова системи у вигляді послідовності збірок (релізів), визначеної відповідно до заздалегідь підготовленим планам і заданими (вже сформульованими) і незмінними вимогами. Сьогодні про інкрементальний підхід найчастіше говорять в контексті поступового нарощування функціональності створюваного продукту.

Може здатися, що індустрія прийшла, нарешті, до загальної «правильної» моделі. Однак каскадна модель, багаторазово «вбита» і теорією, і практикою, продовжує зустрічатися в реальному житті. Спіральна модель є яскравим представником еволюційного погляду, одночасно являє собою єдину модель, що приділяє явну увагу аналізу та попередженню ризиків.

Розглянемо і охарактеризуємо три моделі.

1.4.2. Каскадна (водоспадна) модель

Дана модель (рис. 1.5) припускає строго послідовне (у часі) і однократне виконання всіх фаз проекту з жорстким (детальним) попереднім плануванням в контексті наперед визначених або одного разу і цілком визначених вимог до програмної системи.

Рисунок 1.5 – Каскадна модель життєвого циклу

При активному використанні ця модель продемонструвала свою про блемність в переважній більшості IT-проектів, за винятком, може бути, окремих проектів оновлення програмних систем для крітично-важливих програмно-апаратних комплексів (наприклад, авіоніки чи медичного обладнання). Практика показує, що в реальному світі, особливо у світі бізнес-систем, каскадна модель не повинна застосовуватися. Специфіка таких систем в тому, що вимоги характерізуются високою динамікою коригування та уточнення, неможливо чітко і однозначно визначити вимоги до початку робіт з реалізації (особливо, для нових систем) і швидкої мінливістю вимог у процесі експлуатації системи.

На рис. 1.5 зображені типові фази каскадної моделі життєвого циклу й відповідні активи проекту, що є для одних фаз виходами, а для інших – входами.

В каскадній моделі перехід від однієї фази проекту до іншої предполагає повну коректність результату (виходу) попередньої фази. Однак, наприклад, неточність будь-якої вимоги або некоректна її інтерпретація призводить до того, що доводиться «відкочуватися» до ранньої

фазі проекту, а необхідна переробка не просто вибиває проектну команду з графіка, але призводить до якісного зростання витрат і, не виключено, до припинення проекту в тій формі, в якій він спочатку замислювався. Крім того, ця модель не здатна гарантувати необхідну швидкість відгуку і внесення відповідних змін у відповідь на швидко змінюючі потреби користувачів, для яких програмна система є одним з інструментів виконання бізнес-функцій. І таких прикладів проблем, породжуваних самою природою моделі, можна навести досить багато для відмови від каскадної моделі життєвого циклу.

До основних переваг каскадної моделі відносяться:

- стабільність вимог протягом усього життєвого циклу розробки;
- можливість послідовного усунення виникаючих складнощів;
- визначеність і зрозумілість кроків моделі і простота її застосування;
- спрощення можливості здійснення планування, контролю та управління проектом;
- доступність для розуміння замовниками;
- ефективність для проектів з чіткими і зрозумілими, але важко реалізованими вимогами;
- ефективність для проектів з високими вимогами до якості при відсутності жорстких обмежень витрат і графіка робіт.

Недоліки каскадної моделі життєвого циклу

- складність чіткого формулювання вимог на початку життєвого циклу і неможливість їх динамічної зміни на його протязі;
- послідовність лінійної структури процесу розробки, в результаті повернення до попередніх кроків для вирішення виникаючих проблем призводить до збільшення витрат і порушення графіка робіт;
- непридатність проміжного продукту для використання;
- неможливість гнучкого моделювання систем, що не мають аналогів;
- пізнє виявлення проблем, пов’язаних зі складанням, у зв’язку з одночасною інтеграцією всіх результатів в кінці розробки;
- недостатня участь користувача у створенні системи – тільки на самому початку (при розробці вимог) і в кінці (під час приймальних випробувань);
- неможливість попередньої оцінки якості системи користувачем;
- проблемність фінансування проекту, пов’язана зі складністю одноразової розподілу великих грошових коштів.

Область застосування каскадної моделі.

Обмеження області застосування каскадної моделі визначається її недоліками. Її використання найбільш ефективно в таких випадках:

- при розробці проектів з чіткими, незмінними протягом ЖЦ вимогами, зрозумілими реалізацією і технічними методиками;
- при розробці проекту, орієнтованого на побудову системи або продукту такого ж типу, як вже розроблялися розробниками раніше;
- при розробці проекту, пов’язаного зі створенням і випуском нової версії вже існуючого продукту або системи;
- при розробці проекту, пов’язаного з перенесенням вже існуючого продукту на нову платформу;
- при виконанні великих проектів, в яких задіяно декілька великих команд розробників.

1.4.3. Ітеративна й інкрементальна модель – еволюційний підхід

Ітеративна модель припускає розбивку життєвого циклу проекту на послідовність ітерацій, кожна з яких нагадує «міні-проект», включаючи всі фази життєвого циклу в застосуванні до створення менших фрагментов функціональності (в порівнянні з проектом в цілому).

Мета кожної ітерації – отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим змістом усіх попередніх і поточної ітерації. Результат фінальної ітерації містить всю необхідну функціональність продукту. Таким чином, із завершенням кожної ітерації, продукт розвивається інкрементально.

З точки зору структури життєвого циклу таку модель називають ітеративною (iterative). З точки зору розвитку продукту – інкрементальною (incremental). Досвід індустрії показує, що неможливо розглядати кожен з цих поглядів ізольовано. Найчастіше таку змішану еволюційну модель називають просто ітеративною (говорячи про процес) та/або інкрементальною (говорячи про нарощування функціональності продукту).

Еволюційна модель має на увазі не лише складання працюючої (з погляду результатів тестування) версії системи, але і її розгортання в реальних операційних умовах з аналізом відгуків користувачів для визначення змісту і планування наступної ітерації. «Чиста» інкрементальна модель не передбачає розгортання проміжних збірок (релізів) системи і всі ітерації проводяться по заздалегідь визначеному плану нарощування функціональності, а користувачі (замовник) отримує лише результат фінальної ітерації як повну версію системи.

Таким чином, значимість еволюційного підходу на основі організації ітерацій особливо проявляється у зниженні невизначеності з завершенням кожної ітерації. В свою чергу, зниження невизначеності дозволяє зменшити ризики. Рис. 1.6 ілюструє деякі ідеї еволюційного підходу, припускаючи, що ітеративному розбиттю може бути підданий не тільки життєвий цикл в цілому, що включає перекриваючися фази – формування вимог, проектування, конструювання і т.п., а й кожна фаза може, у свою чергу, розбиватися на уточнюючі ітерації, пов’язані, наприклад, з деталізацією структури декомпозиції проекту – наприклад, архітектури модулів системи.

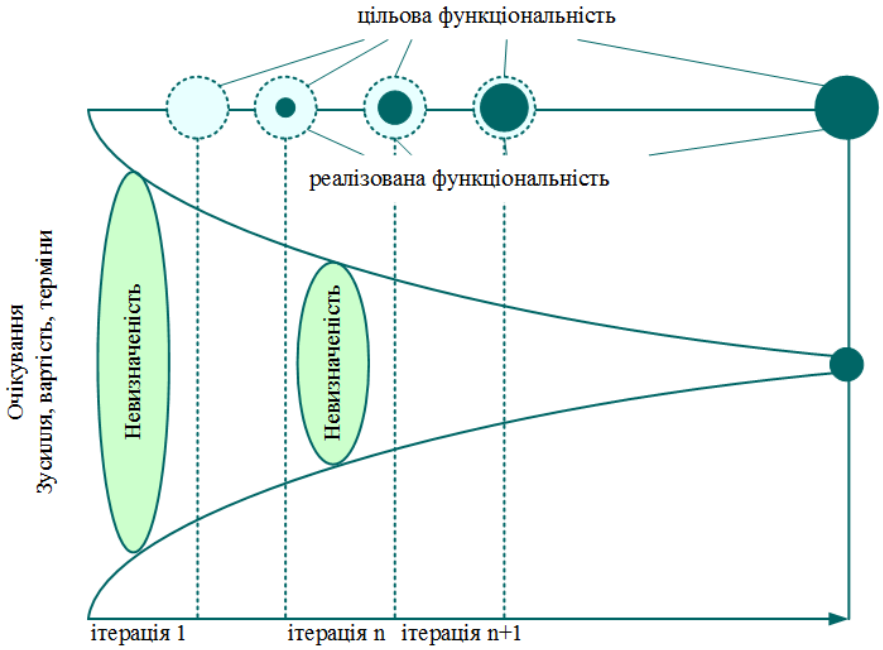


Рисунок 1.6 – Зниження невизначеності та інкрементальне розширення функціональності при ітеративній організації життєвого циклу

Основні переваги итеративної моделі розробки

- зниження ризиків – раннє виявлення конфліктів між вимогами, моделями та реалізацією проекту; велике фокусування на основних завданнях; динамічне формування вимог і управління ними.
- організація ефективного зворотного зв'язку проектної команди зі споживачем, створення продукту, реально відповідає його потребам.
- швидкий випуск мінімально цінного продукту (MVP) і можливість вивести продукт на ринок і почати експлуатацію набагато раніше.

Основні недоліки итеративной моделі розробки:

- Проблеми з архітектурою і накладні витрати – при роботі з хаотичними вимогами і без опрацьованого глобального плану архітектура додатка може постраждати, а на її приведення до адекватного стану можуть знадобитися додаткові ресурси.
- Немає фіксованого бюджету і термінів, а також потрібна сильна залученість замовника в процес – для деяких замовників це неприйнятні умови співпраці з розробником, їм краще підійде водоспадна модель.

Ця модель застосовується для систем, в яких найбільш важливими є функціональні можливості, і які необхідно швидко продемонструвати на CASE-засобах.

1.4.4. Спіральна модель

Найбільш відомим і поширеним варіантом еволюційної моделі є спіральна модель, що стала вже фактично самостійної моделлю, що має різні сценарії розвитку і деталізації.

Спіральна модель (рис. 1.7) була вперше сформульована Баррі Боем в 1988 р Відмінною особливістю цієї моделі є спеціальна увага ризикам, що впливає на організацію життєвого циклу.

Головне досягнення спіральної моделі полягає в тому, що вона пропонує спектр можливостей адаптації вдалих аспектів існуючих моделей процесів життєвого циклу.

Дана модель життєвого циклу характерна при розробці новаторських (нетипових) систем. На початку роботи над проектом у замовника і розробника немає чіткого бачення підсумкового продукту (вимоги не можуть бути чітко визначені) або стовідсоткової впевненості в успішній реалізації проекту (ризики дуже великі). В зв'язку з цим приймається рішення розробки системи по частинах з можливістю зміни вимог або відмови від її подальшого розвитку. Як видно з рис. 1.7, розвиток проекту може бути завершено не тільки після стадії впровадження, але і після стадії аналізу ризику.

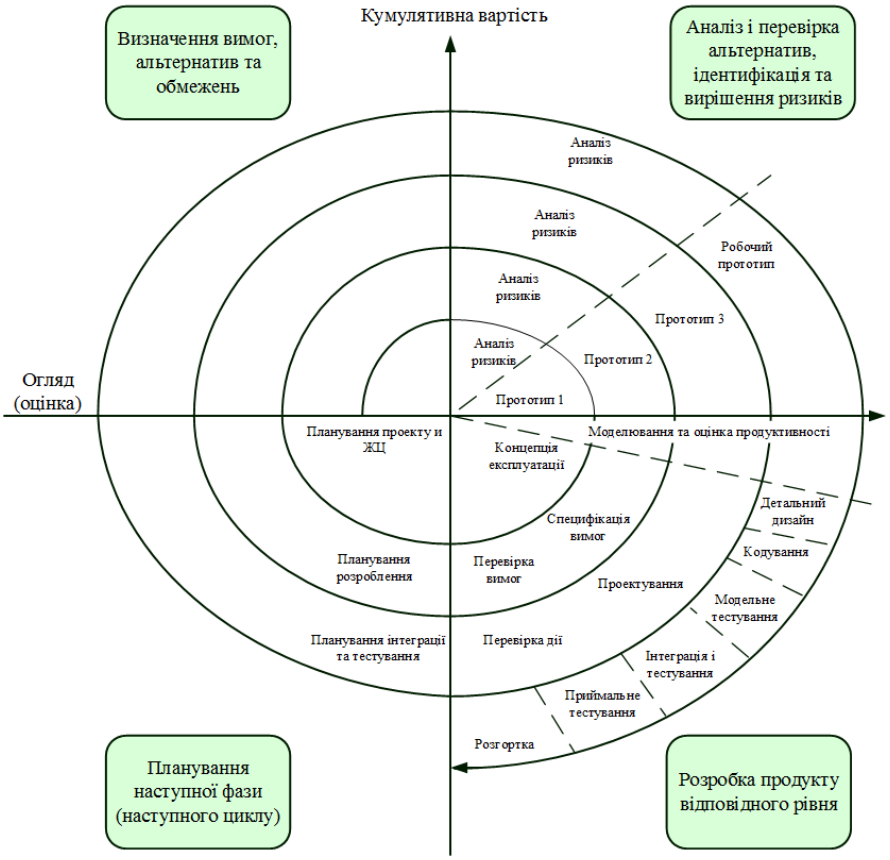


Рисунок 1.7 – Оригінальна спіральна модель ЖЦ, розроблена Боемом

Переваги моделі:

- Дозволяє швидше показати користувачам системи працездатний продукт, тим самим, активізуючи процес уточнення і доповнення вимог;
- Допускає зміну вимог при розробці інформаційної системи, що характерно для більшості розробок, у тому числі і типових;
- Забезпечує більшу гнучкість в управлінні проектом;
- Дозволяє отримати більш надійну і стійку систему. По мірі розвитку системи помилки і слабкі місця виявляються і виправляються на кожній ітерації;
- Дозволяє удосконалювати процес розробки – аналіз, проведений в кожній ітерації, дозволяє проводити оцінку того, що має бути змінено в організації розробки, і поліпшити її на наступній ітерації;
- Зменшуються ризики замовника. Замовник може з мінімальними для себе фінансовими втратами завершити розвиток неперспективного проекту.

Недоліки моделі:

- Збільшується невизначеність у розробника в перспективах розвитку проекту. Цей недолік випливає з попереднього достоїнства моделі;
- Ускладнені операції тимчасового і ресурсного планування всього проекту в цілому. Для вирішення цієї проблеми необхідно ввести тимчасові обмеження на кожну із стадій життєвого циклу. Перехід здійснюється відповідно до плану, навіть якщо не вся запланована робота виконана. План складається на основі статистичних даних, отриманих у попередніх проектах та особистого досвіду розробників.

1.4.5. Сучасні моделі.

Об’єктно-орієнтована модель.

Дана методологія припускає конструювання програмного рішення з готових об’єктів, для яких визначаються правила їх взаємодії, що переводять об’єкти з одного стану в інший. Така модель, що передбачає повну відповідність процесу розробки положенням об’єктно-орієнтованої методології (об’єктно-орієнтований аналіз, проектування, програмування), ефективна у великих проектах, а також там, де застосовуються так звані засоби швидкої розробки (RAD, Rapid Application Development), засновані на цих технологіях і містять готові бібліотеки класів.

Застосовується переважно в дуже великих проектах, де приділяється належна увага етапам аналізу і проектування, а також жорстко контролюється дотримання розробниками встановлених правил.

Моделі швидкої розробки.

Наявність великої кількості формальних процедур і правил істотно звужує свободу дій кожного конкретного програміста, перетворює його на гвинтик у величезній і неповороткою машині. Незважаючи на те що подібні машини здатні цілком успішно справлятися зі стоять перед ними завданнями, зазвичай їх ККД досить низький і питома продуктивність окремого розробника настільки мала, що нормальним може вважатися написання програмістом кількох рядків коду в день.

Кинути виклик подібним перевантаженим формальностями підходам покликані моделі швидкої розробки, такі, як, наприклад, екстремальне програмування. Їх суть полягає у відмові від усього зайвого, що не відноситься безпосередньо до створення якісного програмного

продукту, а за основу беруться лише найбільш ефективні методи створення ПЗ. Особлива увага приділяється питанням взаємодії з замовником, організації продуктивної роботи та тестуванню. Багато ідей швидкої розробки не були чимось новим, наприклад юніт-тести вже давно застосовувалися в багатьох проектах, проте зібрані разом і стали обов'язковими для застосування, вони подіяли позитивний ефект. Про ці методи останнім часом стали говорити все частіше, а їх елементи почали запозичувати багатьма класичними моделями.

У сучасних умовах швидка розробка – це дуже модний підхід, і її використовують все активніше. Основна перевага полягає в тому, що порівняно невеликі групи розробників здатні справлятися з проектами за той же час, який необхідно при застосуванні більш традиційних методів командами на порядок більшої чисельності.

Однак тут є і свої недоліки, зокрема швидка розробка погано підходить для великих проектів і орієнтована в основному на невеликі і середні, крім того, її ефективне використання можливо тільки за умови, що творці ПО мають досить високою кваліфікацією і значним досвідом.

Адаптовані і комбіновані моделі.

Насправді в процесі еволюції моделей життєвого циклу розробки ПЗ нові ідеї не замінювали старі цілком і повністю. Більш правильно вважати, що кожна з них має власну сферу застосування. Крім того, у кожному конкретному випадку може виявитися, що не існує методики, яка ідеально підходить для вирішення даного завдання. У цьому випадку менеджерам програмних проектів варто розглянути варіанти адаптації моделей під конкретні потреби або застосовувати комбіновані методи, що включають елементи різних підходів. Наприклад, успіх швидкої розробки призвів до того, що більш консервативні моделі перейняли найефективніші її прийоми і стали використовувати їх вже в рамках своїх процесів.

Так як поглядів на деталізацію опису життєвого циклу може бути багато, то існують різні методики, серед яких найбільшого поширення набули:

- Rational Unified Process (RUP);
- Enterprise Unified Process (EUP);
- Microsoft Solutions Framework (MSF) в обох поданнях: MSF for Agile і MSF for CMMI (анонсована спочатку як MSF Formal);
- Agile-практики (eXtreme Programming (XP), Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), SCRUM та ін.).

1.5 ІНЖЕНЕРІЯ ВИМОГ

Стадія формування вимог до ПЗ – це найважливіша стадія, оскільки вона визначає успіх усього проекту. Ця стадія складається з таких етапів:

1. Планування робіт включає визначення мети розробки, попередню економічну оцінку проекту, створення плану-графіка виконання робіт, навчання спільної робочої групи;
2. Проведення обстеження діяльності об'єкта (організації) автоматизації, у рамках якого здійснюються: попереднє виявлення вимог до майбутньої системи; визначення структури організації; визначення переліку цілей організації; аналіз розподілу функцій за підрозділами і між співробітниками; виявлення функціональних взаємодій між підрозділами, інформаційних потоків усередині підрозділів і між ними, зовнішніх стосовно організації об'єктів і зовнішніх інформаційних взаємодій; аналіз наявних засобів автоматизації діяльності організації;
3. Побудову моделей діяльності організації, що передбачає обробку матеріалів обстеження;
4. Побудову двох видів моделей:

- моделі "як є", що відображає наявний на момент обстеження стан справ і допомагає зрозуміти, як саме функціонує певне підприємство, а також виявити вузькі місця і сформулювати пропозиції щодо поліпшення ситуації;
- моделі "як має бути", що відображає схему про нові технології роботи підприємства. Кожна з моделей містить повну функціональну й інформаційну модель діяльності організації, а також у разі потреби модель, що описує динаміку поведінки організації.
  - відмовостійкість;
  - кількість клієнтів, що одночасно мають доступ до системи;
  - вимоги безпеки;
  - час очікування відповіді на звернення до системи;
  - виконавські властивості системи (обмеження щодо ресурсів пам'яті, швидкість реакції на звернення до системи тощо).

Наступний крок аналізу вимог – встановлення їх пріоритетності, бо вимоги, висунуті різними носіями інтересів у системі, можуть конфліктувати між собою. Крім того, кожна з вимог потребує для свого втілення певних ресурсів, надання яких може залежати також від визначеного для неї пріоритету.

Ще одним важливим завданням аналізу є передбачення здатності адаптації до можливих змін у вимогах та забезпечення можливостей внесення змін без суттєвого перегляду всієї системи. У процесі аналізу вимог мають бути перевірені їх правдивість та відповідність інтересам замовника.

1.5.1. Автоматизація проектування ІС.

На етапі проектування ІС побажання замовників перетворюються у проектні рішення у формі певної системи програмування.

Проект ІС – це проектно-конструкторська та технологічна документація, в якій подається опис рішень створення та експлуатації ІС у конкретному програмно-технічному середовищі.

В основі проектування будь-якого продукту лежить парадигма подолання складності завдання шляхом його декомпозиції на окремі компоненти.

Технологія проектування ІС – це поєднання методології та інструментальних засобів проектування ІС.

Методологія проектування передбачає наявність концепції, принципів проектування, засобів проектування. Метод проектування ПЗ – це організована сукупність процесів створення моделей, що описують різні аспекти ІС з використанням нотацій. Метод – це сукупність:

- концепцій і теоретичних основ (наприклад, структурний або об'єктно орієнтований підхід);

- нотацій, що використовуються для побудови моделей статичної структури і динаміки поведінки ІС (діаграми потоків даних і діаграми "сутність - зв'язок" для структурного підходу, діаграми варіантів використання, діаграми класів в об'єктно орієнтованому підході);
- процедур, що визначають практичне застосування методу (послідовність і правила побудови моделей, критерії для оцінювання результатів).

Технологія проектування ПЗ - це сукупність технологічних операцій проектування (рис. 1.8) у певній послідовності і взаємозв'язку. Апарат технологічних мереж проектування - це зручний інструмент формалізації технології проектування ІС. Основа його формалізації - визначення технологічної операції проектування у вигляді множини документів (описувач множини фактів), параметрів (описувач одного факту), програм (опис алгоритмів рішення задачі), універсальних множин (повна множина фактів одного типу), на яких задані перетворювачі, ресурси, засоби проектування на конкретному вході/виході.

Рисунок 1.8 - Контекст технологічної операції проектування

Методи реалізуються через конкретні технології і методики, стандарти й інструментальні засоби, що забезпечують виконання процесів ЖЦ ПЗ. Розрізняють методи оригінального проектування, коли створюється оригінальна ІС, та типового проектування, коли ІС компонується з готових типових рішень. Комбінація різних методів проектування зумовлює характер технології проектування ІС. Найвідоміші технології проектування ІС - це канонічна (ручна технологія індивідуального проектування) та індустріальна, що у свою чергу поділяється на автоматизовану (з використанням CASE-технологій) і типову (модельно орієнтовану або параметрично орієнтовану).

Більшість існуючих CASE-засобів засновано на методах структурного або об'єктно орієнтованого аналізу і проектування, що використовує специфікації у вигляді діаграм або текстових описів.

Перехід від моделі яяк є" до моделі "як має бути" може відбуватися двома способами:

1) удосконалюванням діючих технологій на основі оцінки їхньої ефективності;

2) радикальною зміною технологій і перепроектуванням бізнес-процесів.

Стадія проектування включає такі етапи:

- розроблення системного проекту. На цьому етапі дається відповідь на питання: що має робити майбутня ІС?, а саме: визначаються архітектура системи, її функції, зовнішні умови функціонування, інтерфейси й розподіл функцій між користувачами і системою, вимоги до програмних та інформаційних компонентів, склад виконавців і терміни розроблення. Основу системного проекту становлять моделі ІС, що проектуються на основі моделі "як має бути", а результатом діяльності автоматизації є технічне завдання;
- розроблення технічного проекту, яке охоплює проектування системи, що включає проектування архітектури системи і детальне проектування.

Моделі ІС уточнюються і деталізуються до необхідного рівня. На кожній стадії проектування може виконуватися кілька процесів, що визначаються у стандарті ISO/IEC 12207. Кожна програма - це певний перетворювач, поведінку і властивості якого визначають у процесі створення системи так, щоб вирішити певну проблему.

Вимоги до програмної системи - це властивості, які слід мати системі для адекватного виконання своїх функцій.

У сучасних ІТ фаза життєвого циклу, на якій фіксуються вимоги до розроблення програмного забезпечення, визначальна для його якості, термінів та вартості робіт. Саме на цій фазі мають бути зафіксовані реальні потреби користувачів у функціональних, операційних та сервісних можливостях, які має реалізувати розробник. Отже, на цій фазі відбувається домовленість між замовником та виконавцем, яка визначає подальші дії виконавця.

Ціна помилок і нечітких неоднозначних формулювань на цій фазі дуже висока, адже час та засоби витрачаються на непотрібну замовникові програму. Внесення необхідних коректив при цьому може вимагати серйозних переробок, а інколи й повного перепроектування і, відповідно, перепрограмування. За статистичними даними відсоток помилок у постановці завдань перевищує відсоток помилок кодування, і це є наслідком суб'єктивного характеру процесу формулювання вимог та майже повної відсутності засобів його формалізації. Дійовими особами процесу формулювання вимог є:

- носії інтересів замовників (досить часто замовника репрезентують кілька професійних груп, які можуть мати не тільки відмінні, але навіть суперечні потреби);
- оператори, що обслуговують функціонування системи;
- розробники системи.

Процес формулювання вимог складається з двох етапів - збирання та аналізу вимог.

Джерела відомостей про вимоги:

- мета та завдання системи, як їх формулює замовник;
- діюча система або колектив, що виконує її функції;
- загальні знання щодо проблемної галузі замовника;
- відомчі стандарти замовника, що стосуються організаційних вимог, середовища функціонування майбутньої системи, її виконавських та ресурсних можливостей.

Методи збирання вимог:

- інтерв'ю з носіями інтересів замовника та операторами;
- спостереження за роботою діючої системи;
- фіксація сценаріїв усіх можливих випадків використання системи, виконуваних при цьому системою функцій, ролей осіб, що запускають ці сценарії або взаємодіють з системою під час її функціонування.

Множина зібраних вимог може бути розподілена між двома основними категоріями:

1) такі, що відображають можливості, які повинна забезпечити система, - функціональні;

2) такі, що відображають обмеження, пов'язані з функціонуванням системи, - нефункціональні.

Сучасна технологія проектування ПЗ ІС має забезпечувати:

- відповідність стандартові ISO/IEC 12207;
- гарантоване досягнення цілей розробки БС у межах бюджету з дотриманням якості й установленого часу;
- можливість декомпозиції проекту на складові з наступною інтеграцією цих частин;
- мінімальний час одержання працездатного ПЗ ІС;
- незалежність проектних рішень від засобів реалізації 1С (СУБД, операційних систем, мов і систем програмування);
- підтримка CASE-засобів, що забезпечують автоматизацію процесів, виконуваних на всіх стадіях ЖЦ.

Реальне застосування будь-якої технології проектування ПЗ 1С не можливе без розробки стандартів, яких мають дотримуватися всі учасники проекту (це особливо актуально при великій кількості розробників). До них належать стандарти проектування, оформлення проектної документації та інтерфейсу кінцевого користувача із системою.Стандарт проектування встановлює:

а) набір необхідних моделей (діаграм) на кожній стадії проектування і ступінь їх деталізації;

б) правила фіксації проектних рішень на діаграмах, у тому числі правила іменування об'єктів, набір атрибутів для всіх об'єктів і правила їх заповнення на кожній стадії, правила оформлення діаграм тощо;

в) вимоги до конфігурації робочих місць розробників, включаючи настроювання операційної системи та CASE-засобів;

г) механізм забезпечення спільної роботи над проектом, у тому числі правила інтеграції підсистем проекту, правила підтримки проекту в однаковому для всіх розробників стані, правила аналізу проектних рішень на несуперечність.

Стандарт оформлення проектної документації установлює:

а) комплектність, склад і структуру документації на всіх стадіях проектування;

б) вимоги до оформлення документації;

в) правила підготовки, розгляду, узгодження і затвердження документації із зазначенням граничних термінів для кожної стадії;

г) вимоги до засобів підготовки документації;

д) вимоги до настроювання CASE-засобів для забезпечення підготовки документації відповідно до встановлених правил.

Стандарт інтерфейсу користувача із системою регламентує:

а) правила оформлення екранних елементів і елементів управління;

б) правила використання клавіатури і миші;

в) правила оформлення текстів допомоги;

г) перелік стандартних повідомлень;

д) правила обробки реакції користувача.

1.6 ПОВТОРНЕ ВИКОРИСТАННЯ КОМПОНЕНТІВ ІС

Однією з характерних ознак інженерної діяльності є використання готових рішень або деталей. Однак промислове використання готових рішень у програмній інженерії ще не стало повсякденною практикою. Приблизно 80% програмістів працюють над створенням програм обліку й організаційного управління на кількох рівнях: окремого підрозділу фірми, окремого аспекту діяльності фірми, фірми в цілому, корпорації, галузі і, нарешті, держави. Це, переважно, задачі розрахунків, статистики, допомоги у прийнятті рішень при управлінні різноманітними ресурсами - кадровими, фінансовими тощо.

За оцінками експертів, 75% таких робіт дублюють одна одну: на тисячах підприємств створюються програми складського обліку, нарахування зарплати, розрахунку витрат на виробництво продукції, складання маршрутів деталей на виробничому конвеєрі тощо. Хоч більшість із цих програм типові, але кожного разу знаходяться особливості, що не дозволяють застосувати розроблену раніше програму. Тому нині активно розвивається напрямок водночас і науковий, і інженерний, який названо повторним використанням або компонентним розробленням програм.

Компонентне розроблення - це метод побудови ПЗ як композицій готових компонент з конструкцій за каталогом.

Повторне використання - це використання для нових розроблень будь-яких фрагментів інформації, здобутих у процесі розроблення інших ІС.

Повторно використовувані компоненти - елементи знань про минулий досвід розроблення систем програмування, які можна використовувати для створення нових ІС без участі їх розробників.

- 2.1 Поняття архітектури інформаційних систем
- 2.2. Типи архітектур

2.1 ПОНЯТТЯ АРХІТЕКТУРИ ІНФОРМАЦІЙНИХ СИСТЕМ

Рівень розвитку сучасних технологій настільки високий, що дозволяє побудувати інформаційну систему будь-якого масштабу, складності й функціональності. Однак, з огляду на вимоги бізнесу, засновані на показниках різних бізнес-оцінок, виникають додаткові складнооі, вирішення яких зводиться до забезпечення раціонального підходу до процесу проектування, реалізації й подальшій експлуатації інформаційних систем. Виходячи із цього, можна однозначно вважати обрану архітектуру одним з основних показників ефективності створюваної інформаційної системи, а, отже, і успішності бізнесу.

Визначити поняття "архітектура інформаційної системи" можна безліччю способів. Це зв’язано:

1. З відсутністю загальноприйнятого визначення самої інформаційної системи. З огляду на складність структури, достатнім способом описати її можливо тільки при консолідації декількох точок зору, що в кожному конкретному випадку може приводити до різних результатів.
2. З різноманіттям трактувань самого терміну "архітектура".

У результаті, архітектуру інформаційної системи можна описати як концепцію, що визначає модель, структуру, виконувані функції й взаємозв’язок компонентів інформаційної системи.

Процедура вибору архітектури для проектованої інформаційної системи, у ринкових умовах, зводиться до визначення вартості володіння нею. Вартість володіння інформаційною системою складається із планових витрат і вартості ризиків.

Планові витрати містять у собі вартість технічного обслуговування, модернізації, зарплату обслуговуючого персоналу й т.д.

Сукупна вартість ризиків визначається з вартості всіх типів ризиків, їхніх імовірностей і матрицею відповідності між ними. Сама ж матриця відповідності визначається обраною архітектурою інформаційної системи.

Можна виділити найбільш важливі типи ризиків:

- проектні ризики (ризики при створенні системи);
- ризики розробки (помилки, недостатня оптимізація);
- технічні ризики (простої, відмови, втрата даних);
- бізнес-ризики (виникають через технічні ризики й пов’язані з експлуатацією системи);
- невизначеності (пов’язані з варіативністю бізнесів-процесів і складаються з необхідності внесення змін у систему й неоптимальну процедуру функціонування);
- операційні (мають на увазі невиконання набору операцій, можуть виникати через технічні ризики й бути ініціаторами бізнесів-ризиків).

Концепція архітектури інформаційної системи повинна формуватися ще на етапі техніко-економічного обґрунтування й вибиратися такою, щоб вартість володіння нею була мінімальною.

Для того щоб конструктивно визначити архітектуру, необхідно відповісти на ряд питань:

1. Що робить система?
2. На які складові частина вона розділена?
3. Яким чином відбувається взаємодія цих частин?
4. Як і де ці частини розміщені?

Таким чином, можна вважати архітектуру інформаційної системи моделлю, що визначає вартість володіння через наявну в даній системі інфраструктуру.

2.2. ТИПИ АРХІТЕКТУР

Розглядаючи архітектуру великих організацій, прийнято використовувати поняття «корпоративна архітектура». Її можна представити у вигляді сукупності декількох типів архітектур:

- бізнес архітектура (Business architecture);
- IT-архітектура (Information Technology architecture);
- архітектура даних (Data architecture);
- програмна архітектура (Software architecture);
- технічна архітектура (Hardware architecture).

Модель корпоративної архітектури представлена на рис. 2.1.

Рисунок 2.1 – Модель корпоративної інформаційної системи

Технічна архітектура є першим рівнем архітектури інформаційної системи. Вона описує всі апаратні засоби, що використовуються при виконанні заявленого набору функцій, а також включає засобу забезпечення мережної взаємодії й надійності. У технічній архітектурі вказуються периферійні пристрої, мережні комутатори й маршрутизатори, жорсткі диски, оперативна пам'ять, процесори, сполучні кабелі, джерела безперебійного живлення й т.п.

Програмна архітектура являє собою сукупність комп'ютерних програм, призначених для рішення конкретних завдань. Даний тип архітектури призначений для опису додатків, що входять до складу інформаційної системи. На даному рівні описують програмні інтерфейси, компоненти й поведінку.

Архітектура даних поєднує в собі як фізичні сховища даних, так і засоби керування даними. Крім того, до неї входять логічні сховища даних, а при орієнтованості розглянутої компанії на роботу зі знаннями, може бути виділений окремий рівень – архітектура знань (Knowledge architecture). На цьому рівні описуються логічні й фізичні моделі даних, визначаються правила цілісності, складаються обмеження для даних.

Слід особливо виділити рівень IT-архітектури, оскільки він є сполучним.

На ньому формується базовий набір сервісів, які використовуються як на рівні програмної архітектури, так і на рівні архітектури даних. Якщо будь-яка особливість функціонування для цих двох рівнів не була передбачена, то сильно зростає ймовірність збоїв у роботі, а, отже, втрат для бізнесу. У деяких випадках неможливо розділити IT-архітектуру й архітектуру окремого додатка. Таке можливо при високому ступені інтеграції додатків. Прикладом IT-архітектури може служити SharePoint від компанії Microsoft. Цей продукт надає сервіси для спільної роботи й зберігання інформації, що є дуже важливим аспектом функціонування будь-якої компанії. Його базові системні модулі відносяться до IT-архітектури, а користувальницькі – до програмного. Основною функцією IT-архітектури є забезпечення функціонування важливих бізнесів-додатків для досягнення позначених бізнесів-цілей. Якщо деяка функція потрібна відразу в декількох додатках, то її доцільно перенести на рівень IT-архітектури, тим самим підвищивши інтеграцію системи й знизити складність архітектури додатків.

Останнім в ієрархії є рівень бізнес-архітектури або архітектури бізнесів-процесів. На цьому рівні визначаються стратегії ведення бізнесу, способи керування, принципи організації й ключові процеси, що представляють для бізнесу величезну важливість.

2.3. МІКРОАРХІТЕКТУРА Й МАКРОАРХІТЕКТУРА

Терміни мікроархітектура й макроархітектура більшою мірою застосовуються для опису програмних систем. У відповідність із розглянутою моделлю рівнів архітектур корпоративних інформаційних систем, мікроархітектуру можна віднести до рівнів програмної архітектури й архітектури даних, а макроархітектуру – до рівня IT-архітектури.

Мікроархітектура описує внутрішній будову конкретного компонента або підсистеми, а макроархітектура описує будову всієї ІС, як сукупності її компонентів або підсистем.

Складність програмних систем постійно збільшується. Це обумовлено ростом обсягу переданої й оброблюваної інформації, ускладненням самих завдань по обробці інформації й збільшенню кількості таких завдань. Без застосування якого-небудь архітектурного підходу при побудові складних систем, їхнє створення, обслуговування й модифікація, зрештою, стануть нерентабельними для бізнесу.

Для рішення даної проблеми можна використовувати методи абстракції, декомпозиції, інкапсуляції. Так, при розробці програмної системи, що наприклад входить до складу інфраструктури великої організації, вона представляється у вигляді безлічі модулів, кожний з яких виконує певну функцію, а всі разом вони виконують функції самої системи. У цьому випадку, організація кожного модуля буде мікроархітектурою, а способи взаємодії між цими модулями в рамках системи – макроархітектурою.

Існують два принципи, що дозволяють оцінити взаємний вплив компонентів системи один на одного:

- low coupling (слабка зв'язаність);
- high cohesion (сильне зчеплення).

Принцип Low Coupling сприяє розподілу функцій між компонентами системи таким чином, щоб ступінь зв'язаності між ними залишалася низкою. Ступінь зв'язаності (coupling) – це міра взаємозалежності підсистем. Даний принцип пов'язаний з одним з основних принципів системного підходу, що вимагає мінімізації інформаційних потоків між підсистемами.

Підсистема з низьким ступенем зв'язаності (або слабким зв'язуванням) має такі властивості:

- мале число залежностей між підсистемами;
- слабка залежність однієї підсистеми від змін в іншій;
- високий ступінь повторного використання підсистем.

У свою чергу, принцип High Cohesion задає властивість сильного зчеплення всередині підсистеми. У результаті підсистеми виходять сфальцьованими, керованими й зрозумілими.

Зчеплення (функціональне зчеплення) – це міра зв'язаності й сфокусованості функцій підсистеми. Підсистема має високий ступінь зчеплення, якщо її функції тісно зв'язані між собою, і вона не виконує великих обсягів роботи.

Підсистема з низьким ступенем зчеплення виконує безліч різних функцій ніяк не зв'язаних між собою. Такі підсистеми створювати небажано, оскільки вони приводять до виникнення таких проблем:

- труднощі розуміння.
- складність при повторному використанні.
- складність підтримки.
- ненадійність, постійна схильність змінам.

Підсистеми з низьким ступенем зчеплення не мають чіткого функціонального призначення й виконують занадто різнопланові функції, які можна легко розподілити між іншими підсистемами.

Слід відмітити, що зв'язаність є характеристикою системи цілком, а зчеплення характеризує окремо взяту підсистему.

Зв'язаність (coupling) і зчеплення (cohesion) є загальносистемними характеристиками й можуть застосовуватися при проектуванні будь-яких систем.

2.4. АРХІТЕКТУРНИЙ ПІДХІД ДО ПРОЕКТУВАННЯ ІС

Процес проектування інформаційної системи тісним образом пов'язаний з її архітектурним описом, що відбито в деяких визначеннях терміну "архітектура".

Можна виділити п'ять різних підходів до проектування:

- Календарний підхід.
- Підхід, за основу якого взятий процес керування вимогами.
- Підхід, заснований на процесі розробки документації.
- Підхід, в основі якого лежить система керування якістю.
- Архітектурний підхід.

Архітектурний підхід до проектування інформаційних систем можна вважати найбільш зрілим. Його ключовим аспектом є створення фреймворка, тобто каркаса, адаптація якого під потреби конкретної системи буде легко здійсненна. Відповідно до цього, завдання проектування розбивається на дві: розробки багаторазово використовуваного каркаса й створення системи на його основі. Слід зазначити, що дані підзадачі можуть вирішуватися різними групами фахівців. При використанні каркасів з'являється можливість досить швидко змінювати функціональність системи за рахунок ітеративності процесу проектування. Архітектурний підхід покликаний ліквідувати недоліки, що виникають у процесі проектування, заснованому на керуванні вимогами.

2.5 ЗНАЧЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ІНФОРМАЦІЙНИХ СИСТЕМАХ.

Користувачі сучасних інформаційних систем практично завжди взаємодіють із ними за допомогою спеціальних програмних модулів, від показників якості яких залежить рівень якості всієї інформаційної системи цілком.

Існує величезна кількість стандартів для створення правильної й надійної архітектури, а також для розробки й інтеграції програмних систем. Застосування цих стандартів істотно збільшить шанси на успішне створення системи і її подальше безвідмовне функціонування, однак раціональність їхнього застосування повинна визначатися до моменту початку робіт, оскільки складність системи при їхній інтеграції може істотно зрости.

Якістю програмного забезпечення можна вважати сукупність його характеристик, що характеризують можливість задовольняти визначені або умовні потреби всіх зацікавлених осіб.

Можна виділити три аспекти якості:

1. Внутрішня якість (характеристики самого програмного забезпечення).
2. Зовнішня якість (поведінкові характеристики програмного забезпечення).
3. Контекстна якість (відчуття користувачів при різних контекстах використання).

Керуючись цими аспектами, стандарт ISO 9126 виділяє шість характеристик якості програмного забезпечення:

- Функціональність.
- Надійність.
- Продуктивність.
- Зручність використання.
- Зручність супроводу.
- Переносимість.

Функціональність має на увазі здатність ПО вирішувати завдання в певних умовах і поділяється на такі підхарактеристики:

- функціональна придатність (suitability)
- здатність вирішувати потрібний набір завдань;
- точність (accuracy) – здатність одержувати необхідні результати;
- здатність до взаємодії (interoperability) – здатність взаємодії з необхідним набором інших систем;
- захищеність (security) – здатність запобігати неавторизованому доступу до даних і програм;
- відповідність стандартам і правилам (compliance) – відповідність програмного забезпечення різним регламентуючим нормам.

Надійність (reliability) характеризується здатністю програмного забезпечення утримувати функціональність у заданих рамках за певних умов і поділяється на такі підхарактеристики:

- зрілість (maturity) – величина, зворотна частоті відмов програмного забезпечення;
- стійкість до відмов (fault tolerance)
- здатність утримувати певний рівень працездатності при різних відмовах і порушеннях правил взаємодії з оточенням;

- здатність до відновлення (recoverability) – здатність відновлювати необхідний рівень працездатності після відмови;
- відповідність надійності (reliability compliance).

Продуктивність (efficiency) визначається здатністю програмного забезпечення за певних умов гарантувати необхідну працездатність відповідно виділеним для цього ресурсам. Можна також визначити, як відношення одержуваних результатів до витрачених ресурсів. Дана характеристика поділяється на такі підхарактеристики:

- тимчасова з (time behavior) – здатність програмного забезпечення одержувати необхідні результати й забезпечувати передачу необхідного обсягу даних за певний час;
- ефективність використання ресурсів (resource utilization) – здатність програмного забезпечення вирішувати необхідні завдання з використанням заданих обсягів певних видів ресурсів;
- відповідність продуктивності (efficiency compliance).

Зручність використання (usability) характеризується привабливістю для користувачів, зручністю в навчанні й використанні програмного забезпечення. Має такі підхарактеристики:

- зрозумілість (understandability) – величина зворотна зусиллям, витраченим користувачами для усвідомлення застосованості програмного забезпечення для рішення необхідних завдань;
- зручність роботи (operability) – величина зворотна зусиллям, витраченим користувачами, для рішення необхідних завдань за допомогою програмного забезпечення;
- зручність навчання (learnability) – величина зворотна зусиллям, витраченим користувачами, на процес навчання роботі із програмним забезпеченням;
- привабливість (attractiveness) – здатність програмного забезпечення бути привабливим для користувачів;
- відповідність стандартам зручності використання (usability compliance).

Зручність супроводу (maintainability) характеризується зручністю супроводу програмного забезпечення. Дана характеристика також включає ряд підхарактеристик:

- аналізуємість (analyzability) характеризується зручністю проведення аналізу помилок, дефектів, недоліків, необхідності внесення змін й їхніх можливих наслідків;
- зручність внесення змін (changeability) – величина зворотна трудозатратам на виконання необхідних змін;
- стабільність (stability) – величина зворотна ризику появи непередбачуваних наслідків при внесенні необхідних змін;
- зручність перевірки (testability) – величина зворотна необхідним трудозатратам на тестування й інші види перевірок досягнення передбачених результатів при внесенні змін;
- відповідність стандартам зручності супроводу (maintainability compliance).

Переносимість (portability) характеризується здатністю програмного забезпечення зберігати працездатність при зміні організаційних, апаратних і програмних аспектів оточення. Для цієї характеристики виділяються такі підхарактеристики:

- адаптованість (adaptability) – здатність програмного забезпечення без здійснення непередбачуваних дій пристосовуватися до змін оточення;
- зручність установки (installability) – здатність програмного забезпечення встановлюватися в заздалегідь визначене оточення;
- здатність до співіснування (coexistence) – здатність програмного забезпечення функціонувати в загальному оточенні з іншими програмами, розділяючи з ними ресурси;
- зручність заміни (replaceability) – можливість застосування програмного забезпечення замість уже використовуваного для вирішення тих же завдань, у тому ж оточенні;
- відповідність стандартам переносимості (portability compliance).

Всі зазначені характеристики описують внутрішню й зовнішню якість програмного забезпечення. Для опису контекстної якості існує інший, зменшений набір характеристик:

- ефективність (effectiveness) – здатність програмного забезпечення вирішувати користувальницькі завдання із заданою точністю й у заданому контексті;
- продуктивність (productivity) – здатність програмного забезпечення одержувати необхідні результати при використанні заздалегідь визначеної кількості ресурсів;
- безпека (safety) – здатність програмного забезпечення підтримувати необхідний низький рівень ризику завдання збитків людям, бізнесу й навколишньому середовищу;
- задоволеність користувачів (satisfaction) – здатність програмного забезпечення при використанні в певному контексті приносити задоволення користувачам.

Керуючись розглянутими показниками можна значним образом збільшити якість програмних модулів, а, отже, і всієї інформаційної системи в цілому.

2.6 ФУНКЦІОНАЛЬНІ КОМПОНЕНТИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

З огляду на принцип декомпозиції, прийнято проектувати інформаційні системи з поділом функціонального призначення їхніх компонентів, тобто створювати багаторівневе подання.

Можна виділити три основні функціональні групи, призначені для рішення різних за змістом завдань:

1. Взаємодія з користувачами.
2. Бізнес-логіка.
3. Керування ресурсами.

Реалізація такого функціонала відбувається за допомогою створення відповідної програмної системи. Така система також має багаторівневе подання компонентів (рис. 2.2).

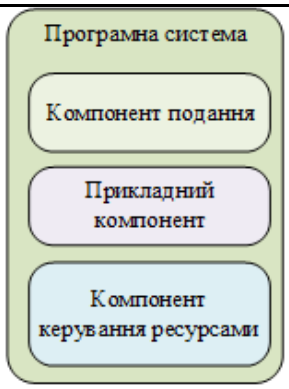


Рисунок 2.2 – Компоненти програмної системи

Компонент подання служить для забезпечення взаємодії користувачів із програмою, тобто обробляє натискання клавіш, руху різних контролерів, здійснює висновок інформації – надає користувальницький інтерфейс.

Прикладний компонент являє собою набір правил й алгоритмів реалізації функцій системи, реакцій на дії користувачів або внутрішніх подій, обробки даних.

Компонент керування ресурсами відповідає за зберігання, модифікацію, вибірку й видалення даних, пов'язаних з розв'язуваним прикладним завданням.

Одним з найважливіших етапів проектування архітектури інформаційної системи є розподіл цих функціональних компонентів по обраній платформній архітектурі. .

2.7 ПЛАТФОРМНІ АРХІТЕКТУРИ ІНФОРМАЦІЙНИХ СИСТЕМ

2.7.1. Напрямки розвитку платформних архітектур.

Можна виділити три напрямки розвитку платформних архітектур:

1. Автономні.
2. Централізовані.
3. Розподілені.

Автономна архітектура має на увазі наявність всіх функціональних компонентів системи на одному фізичному пристрої, наприклад, комп'ютері, й не повинна мати зв'язків із зовнішнім середовищем. Прикладом таких систем можуть служити системні утиліти, текстові редактори й досить прості корпоративні програми. Слід зазначити, що в процесі побудови корпоративної інформаційної системи, як правило, не повинне формуватися не зв'язаних вузлів або модулів. Їхня поява може бути обумовлено певними вимогами до безпеки або надійності.

Централізована архітектура має на увазі виконання всіх необхідних завдань на спеціально відведеному вузлі, потужності якого досить, щоб задовольнити потреби всіх користувачів. Компоненти системи в цьому випадку розподіляються між обчислювальним вузлом, що називається мейнфрейм (mainframe), і термінальною станцією, за якої працює користувач. Термінал містить компонент подання, а мейнфрейм – прикладний компонент і компонент керування ресурсами. Слід зазначити, що термінал виступає винятково у вигляді пристрою виводу й не має інших функціональних можливостей.

До переваг такої архітектури можна віднести:

- відсутність необхідності адміністрування робочих місць;
- легкість обслуговування й експлуатації системи, оскільки всі ресурси зосереджені в одному місці.

Недоліками подібної архітектури є:

- функціонування всієї системи повністю залежить від головного вузла (мейнфрейма);
- всі ресурси й програмні засоби є колективними й не можуть бути змінені під потрібні конкретних користувачів.

Щоб позбутися від останнього недоліку, у сучасних інформаційних системах застосовуються технології віртуалізації, завдяки чому стає можливим виділити кожному користувачеві необхідну кількість ресурсів й установити необхідне програмне забезпечення.

Слід відмітити, що за допомогою технологій віртуалізації можна створити практично будь-яку архітектуру, використовуючи при цьому тільки ресурси мейнфрейма.

Поява й розвиток розподілених архітектур пов'язані з інтенсивним розвитком технічних і програмних засобів. У даному типі архітектури функціональні компоненти інформаційної системи розподіляються по наявних вузлах залежно від поставлених цілей і завдань. Можна виділити шість основних характеристик архітектури розподілених систем:

- спільне використання ресурсів (як апаратних, так і програмних);
- відкритість – можливість збільшення типів і кількості ресурсів;
- паралельність – можливість виконання декількох процесів на різних вузлах системи (при цьому вони можуть взаємодіяти);
- масштабованість – можливість додавати нові властивості й методи;
- відмовостійкість – здатність системи підтримувати часткову функціональність за рахунок можливості дублювання інформації, апаратної і програмної складових.

До недоліків розподілених систем варто віднести:

- структурна складність;
- складно забезпечити достатній рівень безпеки;
- на керування системою потрібне велика кількість зусиль;
- непередбачена реакція на зміни.

Всі вони пов’язані в першу чергу зі складною структурою, різноплановим устаткуванням і складною системою розподілу прав доступу. Необхідно враховувати всі з них, інакше розроблена інформаційна система не зможе функціонувати в рамках очікуваних параметрів.

2.7.2. Види розподілених архітектур.

Існують такі види розподілених архітектур:

- архітектура «файл-сервер»;
- архітектура «клієнт-сервер»;
- архітектура web-додатків.

Файл-серверна архітектура має на увазі наявність виділеного мережного ресурсу для зберігання даних. Такий ресурс називається «файловим сервером». При такій архітектурі всі функціональні компоненти системи розташовані на користувальницькому комп’ютері, що називається «клієнтом», а самі дані перебувають на сервері.

Така організація системи має такі переваги:

- багатокористувальницький режим режим роботи з даними, що зберігаються на сервері;
- централізоване керування правами доступу до загальних даних;
- низька вартість розробки;
- висока швидкість розробки.

Недоліки файл-серверної архітектури:

- послідовний доступ до загальним даних і відсутність гарантії їхньої цілісності;
- продуктивність (залежить від продуктивності мережі, клієнта й сервера);

Класичне подання файл-серверної архітектури представлене на рис. 2.3.

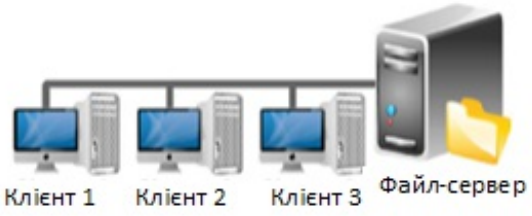


Рисунок 2.3 – Подання «файл-серверної» архітектури

Архітектура «клієнт-сервер» являє собою мережеву інфраструктуру, в якій сервери є постачальниками певних сервісів (послуг), а клієнтські комп’ютери виступають їхніми споживачами.

Класичне подання клієнт-серверної архітектури має на увазі наявність у мережі сервера й декількох підключених до нього клієнтів. У таких системах сервер, в основному, відіграє роль постачальника послуг з використання бази даних.

Ця архітектурна модель називається дворівневою (two-tier architecture) і подана на рис. 2.4.

Рисунок 2.4 – Подання «файл-серверної» архітектури

Переваги даної архітектури:

- підтримка багатокористувальницької роботи;
- гарантія цілісності даних;
- наявність механізмів керування правами доступу до ресурсів сервера;
- можливість розподілу функцій між вузлами мережі.

Недоліки:

- вихід з ладу сервера може спричинити непрацездатність всієї системи;
- потреба високого рівня технічного персоналу;
- висока вартість устаткування.

При збільшенні масштабів системи може знадобитися заміна апаратної частини сервера й клієнтських машин. Однак, при збільшенні числа користувачів виникає необхідність синхронізації версій великої кількості додатків. Для рішення цієї проблеми використовують багатоланкові архітектури (три й більше рівні). Частина загальних додатків переноситься на спеціально виділений сервер, тим самим знижуються вимоги до продуктивності клієнтських машин. Клієнти з низькою обчислювальною потужністю називають «тонкими клієнтами», а з високою продуктивністю – «товстими клієнтами». При багатоланковій архітектурі з виділеним сервером додатків існує можливість використання портативних пристроїв. Багатоланкова архітектура показана на рис. 2.5.

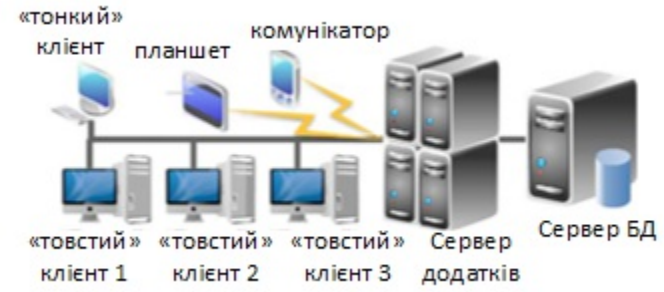


Рисунок 2.5 – Приклад багатоланкової «клієнт-серверної» архітектури

Використання такого типу архітектури обумовлено високими вимогами додатка до ресурсів. У такому випадку існує сенс винести його на окремий сервер й, тим самим, знизити вимоги до продуктивності робочих станцій.

Грамотний підбір характеристик сервера додатків, сервера баз даних і клієнтських робочих станцій дозволить створити інформаційну систему із прийнятною вартістю володіння.

Слід відмітити, що розподіл функціональних компонентів системи при використанні клієнт-серверної архітектури може виконуватися декількома способами, зображеними на рис. 2.6.

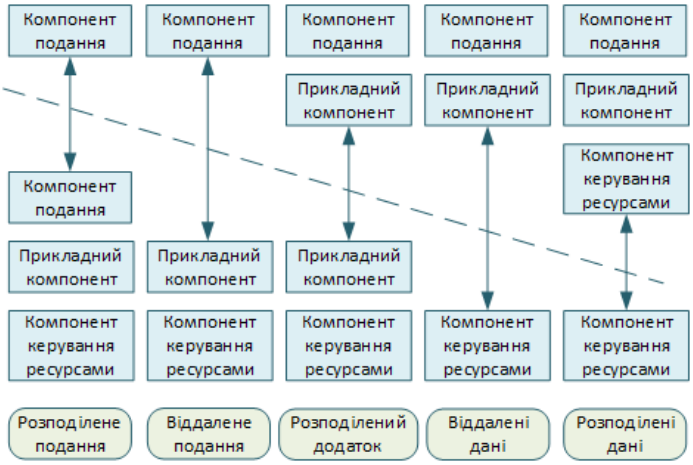


Рисунок 2.6 – Приклади розподілу функціональних компонентів

Архітектура web-додатків або архітектура web-сервісів має на увазі надання деякого сервісу, доступного в мережі Internet, через спеціальний додаток.

Основою для надання таких послуг служать відкриті стандарти й протоколи SOAP, UDDI й WSDL. SOAP (Simple Object Access Protocol) визначає формат запитів до web-сервісів. Дані між клієнтом і сервісом передаються в SOAP-конвертах (envelops). WSDL (Web Service Description Language) служить для опису інтерфейсу надаваного сервісу. Перед розгортанням web-додатка потрібно скласти його опис, указати адреса, список підтримуваних протоколів, перелік припустимих операцій, а так само формати запитів і відповідей. UDDI (Universal Description, Discovery and Integration) являє собою протокол пошуку web-сервісів у мережі Internet. Пошук здійснюється за їхніми описами, які розташовані в спеціальному реєстрі.

Архітектура таких сервісів схожа за концепцією з багатоланкової клієнт-серверною, однак, сервера додатків і баз даних розташовуються в мережі Internet.

Можна виділити три технології, які можливо використати для побудови розподіленої архітектури web-сервісу:

- EJB (Enterprise JavaBeans).
- DCOM (Distributed Component Object Model).
- CORBA (The Common Object Request Broker Architecture).

Ідеєю для створення EJB було бажання створити інфраструктуру для легкого додавання й видалення компонентів зі зміною функціональності сервера. EJB дозволяє розроблювачам створювати власні додатки із заздалегідь створених модулів. При цьому можлива їхня зміна, що робить процес розробки гнучким і набагато більше швидким. Дана технологія сумісна з CORBA й Java API.

Взаємодія між клієнтів і сервером у цьому випадку представляється як взаємодія EJB-об’єкта, що генерується спеціальним генератором, і EJB-компонента, написаного розроблювачем. При необхідності викликати метод в EJB-компонента, що перебуває на сервері, викликається однойменний метод EJB-об’єкта, розташованого на стороні клієнта, що зв’язується з необхідним компонентом і викликає необхідний метод.

Переваги EJB:

- просте й швидке створення;
- Java-оптимізація;
- кросплатформність;
- вбудована безпека.

Недоліки EJB:

- складність інтегрування з додатками;
- погана масштабованість;
- низька продуктивність;
- відсутність міжнародної стандартизації.

DCOM являє собою розподілену програмну архітектуру від компанії Microsoft. З її допомогою програмний компонент одного комп’ютера може передавати повідомлення програмному компоненту іншого комп’ютера, причому з’єднання встановлюється автоматично. Для надійної

роботи потрібно забезпечити захищене з'єднання між зв'язаними компонентами, а також створити систему перерозподіл трафіку.

Переваги DCOM:

- незалежність від мови;
- динамічне знаходження об'єктів;
- масштабованість;
- відкритий стандарт.

Недоліки DCOM:

- складність реалізації;
- залежність від платформи;
- пошук через службу Active Directory;
- відсутність іменування сервісів через URL.

Технологія CORBA розглядає всі додатки в розподіленій системі як набір об'єктів. Об'єкти можуть одночасно виступати в ролі клієнта й сервера, викликаючи методи інших об'єктів і відповідаючи на їхні виклики. Застосування даної технології дозволяє будувати системи, що перевершують по складності й гнучкості системи з архітектурою клієнт-сервер (як дворівневої, так і трирівневої).

Переваги CORBA:

- незалежність від платформи;
- незалежність від мови;
- динамічні виклики;
- динамічне виявлення об'єктів;
- масштабованість;
- індустріальна підтримка.

Недоліки:

- відсутність іменування по URL;
- практично повна відсутність реалізації CORBA-сервісів;

При грамотному підході до побудови архітектури інформаційної системи може знадобитися використання відразу декількох з розглянутих технологій. Кожна з них буде реалізовувати певний функціонал, що може зажадати також декількох типів платформних архітектур. В одній системі можуть працювати кілька файлів-серверів, кілька серверів додатків і кілька серверів баз даних. У такий спосіб можна розподіляти навантаження в системі або групувати набори сервісів відповідно виконуваним функціях.

2.8 ПОНЯТТЯ Й КЛАСИФІКАЦІЯ АРХІТЕКТУРНИХ СТИЛІВ

Більша частина процесів по проектуванню інформаційних систем має на увазі використання досвіду реалізації схожих проектів. Складно уявити систему, для реалізації якої не можна було б застосувати вже готові рішення або досвід, отриманий при їхньому створенні. Архітектурний стиль можна охарактеризувати як подібність у підходах до реалізації поставлених завдань, обумовлене досвідом. Він визначає перелік компонентів системи, способи й умови їхньої взаємодії. На жаль, всупереч безлічі спроб, не існує стандартних мов опису архітектур.

Архітектурні стилі поділяються на п'ять груп (рис. 2.7):

- Потоки даних (Data Flow Systems).
- Виклик з поверненням (Call-and-Return Systems).
- Незалежні компоненти (Independent Component Systems).
- Централізовані дані (Data-Centric Systems).
- Віртуальні машини (Virtual machines).

Системи потоків даних, у свою чергу, поділяються на:

- системи пакетно-послідовної обробки (Batch Sequential Systems);
- системи типу конвеєри й фільтри (Pipe and Filter Architecture).

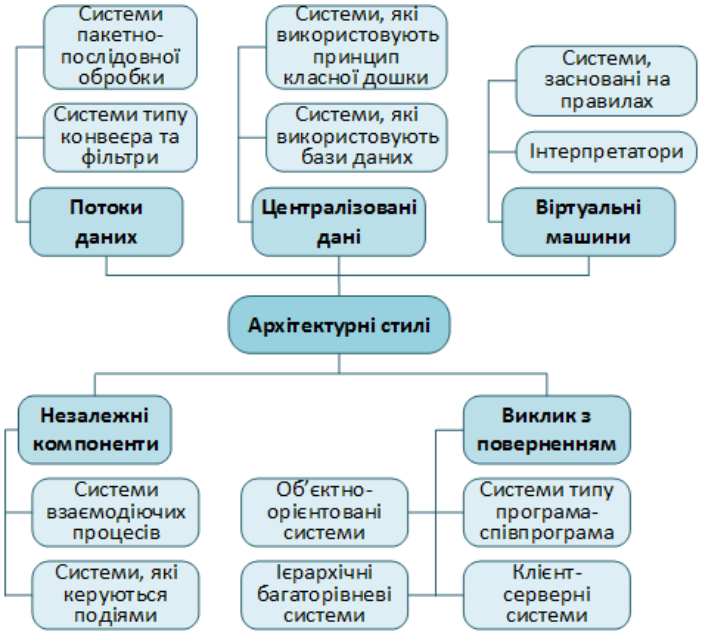


Рисунок 2.7 – Класифікація архітектурних стилів

У системах пакетно-послідовної обробки розв'язуване завдання ділиться на сукупність підзадач, механізм рішення яких буде реалізований в окремих програмних модулях, об'єднаних у лінійну структуру. Вихідні даний однієї підзадачі є вхідними даними для іншої.

Стиль "конвеєри й фільтри" може вважатися узагальненням пакетно-послідовної обробки. Його структура складається з безлічі модулів, кожний з яких виконує один або кілька процесів. Результати виконання одного процесу можуть передаватися як одному, так і декільком модулям, причому різними способами. Такі системи реалізують принцип конвеєра, у якому можуть бути присутнім зворотні зв'язки.

Гарним прикладом такого підходу може служити компілятор, що послідовно виконує лексичний аналіз, семантичний аналіз, оптимізацію й генерацію коду.

Системи, що функціонують за допомогою викликів з поверненнями є синхронними програмними архітектурами, клієнтська частина яких припиняє функціонування на час обслуговування власного запиту сервером. Такі архітектури можуть включати довільну кількість рівнів вкладеності. Існує кілька типів подібного роду систем:

- програма-співпрограма (Main Programm and Subroutines);
- об'єктно-орієнтовані системи (Object-Oriented Systems);
- клієнт-серверні системи (Client-Server Systems);
- ієрархічні багаторівневі системи (Hierarchically Layered Systems).

Стиль «програма-співпрограма» є реалізацією ідей структурного програмування й має на увазі наявність головної керуючої програми (контролера), відповідальної за процес функціонування, і безлічі співпрограм, що реалізують функціональність. Різновидом даного підходу вважається архітектура "ведучий- ведений" (Master-Slave Architecture), у якій основна програма й співпрограми працюють одночасно (паралельно). Контролер виконує функції диспетчера процесу, у той час, як співпрограми виконують завдання, по завершенні яких запитують у нього нові.

Об'єктно-орієнтовані системи є окремим випадком систем «програма-співпрограма». Спілкування між об'єктами, що включають в собі код і дані, здійснюється або за допомогою викликів процедур, або за допомогою повідомлень. Слід відмітити, що викликаючий об'єкт повинен знати, де перебуває викликуваний, крім того, йому необхідно знати набір інтерфейсів, які він може використати. У результаті інкапсуляції приховуються дані про реалізації якого-небудь об'єкта, у результаті чого стає можливим вносити в нього зміни без повідомлення кінцевих користувачів, що є незаперечною перевагою даного типу архітектури. Також до переваг можна віднести природну підтримку розпаралелювання процесів.

Клієнт-серверні системи також можна вважати окремим випадком стилю «програма-співпрограма», з тією лише різницею, що контролер і співпрограми можуть розташовуватися на різних вузлах мережі.

Для великомасштабних систем застосовують ієрархічно багаторівневий стиль, в якому кожний з наявних шарів можна розглядати як набір серверів для вищого шару. Відповідно, вищий шар є клієнтом, а нижній – сервером. Такий стиль виправдано використовується для створення стеків протоколів або операційних систем. Головною його перевагою є ведення розробки кожного із шарів незалежно. Слід відмітити, що не всі алгоритми можна реалізувати у вигляді багаторівневої структури, тому її використання не завжди виправдане.

Системи, що функціонують за принципом незалежних компонентів, використовують механізм неявного виклику операторів, тобто взаємодіючі оператори можуть працювати незалежно й розташовуватися на різних хостах мережі. Виділяють два типи подібних систем:

- системи взаємодіючих процесів (Communicating Sequential Processes);
- системи, керовані подіями (Event- Based Systems).

Основним принципом функціонування систем взаємодії є обмін повідомленнями між незалежними процесами.

У системах, керованих подіями, процеси запускаються тільки в момент появи певної події, однак одержувач повідомлення про подію може не знати про відправника, а відправник – про одержувача. Схожі принципи функціонування у систем з перериваннями.

При наявності в системі загальнодоступного централізованого сховища інформації, її відносять до стилю централізованих даних (репозиторія). При використанні даного підходу дані вводяться в системи однократно й при необхідності доповнюються. Це забезпечує загальний доступ декількох додатків до даних, можливість обміну даними, виключає дублювання й

спрощує масштабування. Існує два різновиди подібних систем:

- системи, засновані на використанні централізованої бази даних (Database Systems) найчастіше мають на увазі наявність реляційної бази даних;
- системи, що використовують принцип класної дошки (Blackboard Systems).

Системи, побудовані за принципом класної дошки, характеризуються наявністю загальної розділеної пам'яті (бази даних), що зберігає результати виконуваних процесами дій. У таких системах існує можливість оповіщення зацікавлених процесів про зміни в необхідній їм інформації.

Віртуальні машини є спеціальними емуляторами, що забезпечують програмний інтерфейс відмінний від поточного. Віртуальні машини можуть прямо працювати з апаратною платформою або бути надбудовами операційної системи. При розгляді інформаційної системи у вигляді багатошарової структури, віртуальна машина буде верхнім шаром, що забезпечує взаємодію з користувальницькими додатками. Вони поділяються на:

- інтерпретатори (Interpreters);
- системи, засновані на правилах (Rule- Based Systems).

Інтерпретатори призначені для забезпечення працездатності різного роду програм, створених для різних платформ. Наприклад, запуск і налагодження Linux- додатків у середовищі Windows.

Системи, засновані на правилах, представляють всі дані й логіку у вигляді набору спеціалізованих правил. У таких системах для кожного завдання існує набір фактів і набір правил. Для рішення завдання до фактів застосовуються правила доти, поки не буде отриманий результат. Прикладом таких систем може бути CLIPS.

Доцільність використання різних архітектурних стилів наведена в таблиці 2.1.

Таблиця 2.1 – Доцільність використання стилів

Назва стилю	Доцільність використання
Пакетно-послідовний	Розв'язуване завдання можна поділити на підзадачі, які використовують єдину операцію вводу-виводу
Конвеєри й фільтри	Процес рішення завдання можна представити у вигляді послідовності повторюваних операцій над незалежними однотипними даними
Програма-співпрограма	Фіксований порядок операцій, простоювання через очікування відповідей від компонентів
Об'єктно-орієнтований	Можливість використання механізмів спадкування, розташування об'єктів на різних хостах
Клієнт-серверний	Можливість представити розв'язуване завдання у вигляді набору запитів і відповідей від клієнтів до сервера
Ієрархічний багаторівневий	Можливість представити завдання як сукупність шарів з певними інтерфейсами, необхідність у різних варіантах реалізації бізнеси-логіки, використання наявних реалізацій
Взаємодіючі процеси	Механізм взаємодії між процесами – обмін повідомленнями, обсяг довгострокових централізованих даних невеликий
Керовані події	Асинхронний процес функціонування системи, можливе подання системи у вигляді незалежних процесів
Централізована база даних	Доступна СУБД, завдання розділяються на виробників і споживачів даних, черговість виконання компонентів визначається послідовністю вхідних запитів
Класна дошка	Велика кількість клієнтів спілкуючихся між собою
Інтерпретатор	Необхідно нівелювати специфіку платформ, надати специфічне середовище роботи
Заснований на правилах	Рішення завдання можна представити у вигляді набору правил й умов їхнього застосування

2.9. ФРЕЙМВОРКИ (КАРКАСИ)

Термін фреймворк можна визначити як загальноприйняті архітектурно-структурні рішення й підходи до проектування. Можна сказати, що фреймворк являє собою загальне рішення складного завдання.

Класифікація фреймворків показана на рис.2.8.

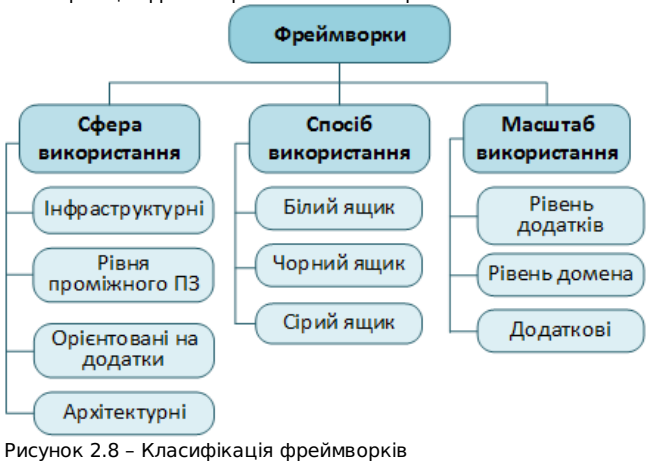


Рисунок 2.8 – Класифікація фреймворків

Інфраструктурні фреймворки (System Infrastructure Frameworks) спрощують процес розробки інфраструктурних елементів, застосовуються усередині організації й не продаються.

Фреймворки рівня проміжного програмного забезпечення (Middleware Frameworks)

застосовуються для вбудовування додатків і компонентів.

Фреймворки, орієнтовані на додатки, використовуються для підтримки систем, орієнтованих на роботу з кінцевими користувачами в конкретній предметній області.

У відповідності зі стандартом КОЛЕС 42010 архітектурний фреймворк визначається як «сукупність угод, принципів і практик, які використовуються для опису архітектур і прийнятих відповідно деякому предметному домену й (або) у співтоваристві фахівців (зацікавлених осіб)». Архітектурний фреймворк містить у собі опис зацікавлених осіб, типові проблеми предметної області, архітектурні точки зору й методи їхньої інтеграції.

Фреймворки, використовувані за принципом білого ящика (Architecture-driveng framework), застосовують методи спадкування й динамічного зв'язування для формування основних елементів додатка. Такі фреймворки визначаються через інтерфейси об'єктів, що додають у систему. Для роботи з ними необхідна докладна інформація про класи, розширення яких необхідно.

Фреймворки, що функціонують за принципом чорного ящика, також називають фреймворками, керованими даними. Основними механізмами формування додатків, у цьому випадку, виступають композиція й параметризація, при цьому функціональність забезпечується додаванням додаткових компонентів. Слід зазначити, що процес використання фреймворків, що працюють за принципом чорного ящика простіше, ніж працюючих за принципом білого ящика, однак їхня розробка складніше.

На практиці застосовують підхід сірого ящика (grey box), що є комбінацією обох підходів.

Фреймворки рівня додатків (application frameworks) надають функціонал по реалізації типових додатків (GUI, бази даних і т.д).

Фреймворки рівня домену (Domain Frameworks) застосовуються для створення додатків у певній предметній області. Їхня класифікація представлена на рис. 2.10.

М'які фреймворки мають на увазі можливість власного настроювання для вирішення конкретного завдання, у той час як тверді – ні.

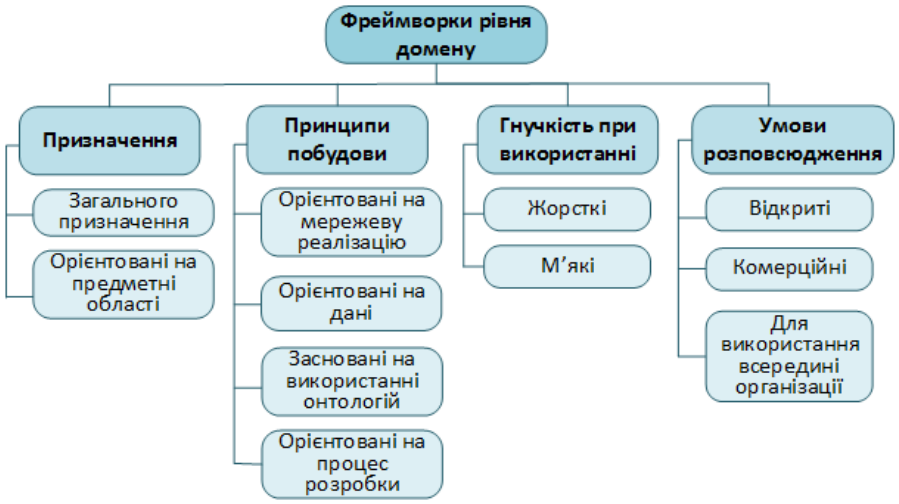


Рисунок 2.10 – Класифікація фреймворків рівня домену

Допоміжні фреймворки (Support Frameworks) застосовуються для рішення приватних завдань.

У світі існує величезна кількість різних фреймворків. Як приклади можна виділити п'ять найбільш відомих:

1. Фреймворк Захмана.
2. TOGAF.
3. DoDAF.
4. FEA.
5. Gärtner.

2.9.1. Фреймворк Захмана.

Фреймворк Захмана є одним із самих старих архітектурних фреймворків. Він був створений співробітником компанії IBM Джоном Захманом (John Zachman). Захман заклав в основу свого фреймворка класифікацію (таксономію) артефактів системи. Серед них можна виділити дані, функціональність, моделі, специфікації й документи. У результаті, можна вважати цей фреймворк онтологією верхнього рівня, що описує конкретну систему (таблиця з).

Для побудови таксономії Захманом запропоновано відповісти на шість питань про функціонування організації: що, як, де, хто, чому. Дані питання ставляться до наступних аспектів системи:

- використовувані дані (що?);
- процеси й функції (як?);
- місця виконання процесів (де?);
- організації й персоналії (хто?);
- керуючої події (коли?);
- мети й обмеження, що визначають роботу системи (чому?).

	Використовувані дані (що?)	Процеси й функції (як?)	Місця виконання процесів (де?)	Організації й персоналії (хто?)	Керуючі події (коли?)	Мети й обмеження	
Контекст	Список основних сутностей	Основні бізнес-процеси	Територіальне розміщення організації	Важливі зовнішні організації	Список подій	Бізнес-стратегія	АНАЛИТИКИ
Бізнес-модель	Відносини між сутностями	Докладний опис бізнес- процесів	Система логістики	Модель потоків даних	Базовий графік робіт	Дерево цілей, Бізнес-план	Топ-менеджери
Системна	Концептуальні	Архітектура	Архітектура	Інтерфейси	Модель роботи	Бізнес	

Системна модель	Концептуальні моделі даних	Архітектура додатків	розподіленої системи	інтерфейси користувача	Модель роботи з подіями	Бізнес-правила	Архітектори
Технологічна модель	Фізична модель даних	Програмно-апаратна архітектура	Технологічна архітектура	Архітектура подання	Алгоритми обробки подій	Правила обробки подій	Розробник
Детальний опис	Специфікації форматів даних	Виконуваний код	Архітектура мережі	Ролі й права користувачів	Обробка подій за допомогою переривань	Алгоритми роботи системи	Адміністратори
Функціонуюча організація	Дані	Реалізована функціональність	Функціонуюча мережна інфраструктура	Організаційна структура організації	Історія функціонування системи	Реалізовані стратегії	Користувачі

Відповідати на ці питання необхідно з різним ступенем деталізації. Описано шість рівнів:

- рівень контексту;
- рівень бізнесів-описів;
- системний рівень;
- технологічний рівень;
- технічний рівень;
- рівень реальної системи.

Стосовно до кожного рівня деталізації існує своя зацікавлена особа (stakeholders), тобто точка зору:

- аналітики (рівень контексту);
- топ-менеджери (рівень бізнес- описів);
- архітектори (системний рівень);
- розроблювачі (технологічний рівень);
- адміністратори (технічний рівень);
- користувачі (рівень реальної системи).

За результатами виконаних дій формується матриця розміром 6х6, у кожній комірці якої розташовуються артефакти. Для заповнення комірок уведені наступні правила:
Стовпчика можна міняти місцями, але не можна додавати й видаляти.

- Кожному стовпчику відповідає власна модель.
- Кожна з моделей має бути унікальна.
- Кожен рівень (рядок) є описом системи з погляду групи користувачів (представляє окремий вид).
- Кожна з комірок унікальна.
- Кожна комірка містить опис аспекту реалізації системи у вигляді моделі й текстового документа.
- Заповнення комірок повинне відбуватись послідовно зверху вниз.

Перший рядок матриці визначає контекст всіх інших й являє собою загальний погляд на організацію. Другий рядок описує функціонування організації в бізнесах-термінах. Третій рядок описує бізнес- процеси в термінах інформаційних систем. Четвертий рядок дозволяється розподілити дані й виконувані над ними операції між конкретними апаратними і програмними платформами. П'ятий рядок описує конкретні моделі устаткування, мережеві топології й програмний код. Шостий рядок описує готову систему у вигляді інструкції користувачів, довідкових баз даних і т.д.

Стовпець «Використовувані дані», відповідно до рівнів, містить у своїх комірках такі артефакти: 1 рівень – список основних сутностей; 2 рівень – семантична модель; 3 рівень – нормалізована модель; 4 рівень – фізична модель даних або ієрархія класів; 5 рівень – опис моделі мовою керування даними; 6 рівень – фактичні набори даних, статистики й т.д.

Стовпець «Процеси й функції» описує порядок дій для деталізації процесу переходу від місії організації до опису конкретних операцій: 1 рівень – перерахування ключових бізнесів-процесів; 2 рівень – опис бізнес-процесів; 3 рівень – опис операцій над даними й архітектури додатків; 4 рівень – методи класів; 5 рівень – програмний код; 6 рівень – виконувані модулі. Із четвертого рівня розгляд ведеться в рамках окремих додатків.

Стовпець «Місця виконання процесів» визначає розташування компонентів системи й мережеву інфраструктуру: 1 рівень – розташування основних об'єктів; 2 рівень – модель взаємодії об'єктів; 3 – розподіл компонентів інформаційної системи за вузлами мережі; 4 рівень – фізична реалізація на апаратних і програмних платформах; 5 рівень – використовувані протоколи й специфікації каналів зв'язку; 6 – опис функціонування мережі.

Стовпець «Організації й персоналії» визначає учасників процесу функціонування: 1 рівень – список партнерів, підрозділів організації, виконувані функції; 2 рівень – повна організаційна діаграма (можуть бути присутнім вимоги до інформаційної безпеки); 3 рівень – учасники бізнесів-процесів й їхні ролі; 4 рівень – вимоги до користувальницьких інтерфейсів; 5 рівень – правила доступу до об'єктів; 6 рівень – фізична реалізація в коді.

Стовпець «Керуюча подія» визначає тимчасові параметри системи й бізнесів-процесів: 1 рівень – список значимих для системи подій; 2 рівень – базовий графік робіт; 3 рівень – моделі роботи з подіями; 4 рівень – алгоритми обробки подій; 5 рівень – програмна реалізація; 6 рівень – історія функціонування системи.

Стовпець «Мета й обмеження» вказує послідовність дій для переходу від завдань бізнесу до вимог для елементів системи: 1 рівень – бізнес-стратегія; 2 рівень – дерево цілей і бізнес-план; 3 рівень – правила й обмеження для бізнесів-процесів; 4 рівень – додатки, що включаються до складу системи; 5 рівень – алгоритми роботи додатків; 6 рівень – фізична реалізація в коді.

За допомогою фреймворку Захмана не можна описати динаміку поводження системи (розвиток), крім того, у ньому не існує механізму контролю за змінами. Даний фреймворк розповсюджується на комерційній основі.

2.9.2. ФреймворкTOGAF.

Фреймворк TOGAF (The Open GroupArchitecture Framework) являє собою набір засобів для розробки архітектур різного призначення. З його допомогою інформаційна система подається як сукупність модулів.

В рамках TOGAF дається особливе визначення архітектури – «формальний опис системи, або детальний план системи на рівні компонентів і методології їх реалізації». Загальноприйняте ж

визначення архітектури (відповідно до стандарту ANSI / IEEE 1471-2000) визначається як «опис організації системи в термінах компонентів, їх взаємозв'язків між собою і з навколишнім середовищем і принципи управління їх розробкою і розвитком».

TOGAF складається з чотирьох архітектурних доменів:

- бізнес-архітектура (описує ключові бізнес-процеси, стратегію розвитку бізнесу і принципи управління);
- архітектура рівня додатків (описує інтерфейси додатків і способи їх застосування в термінах бізнес-сервісів);
- архітектура рівня даних (визначає логічну й фізичну структуру даних в організації);
- технологічна архітектура (визначає програмну, апаратну і мережеву інфраструктури).

Головними складовими частинами TOGAF є:

- ADM-методика (Architecture Development Method), що описує процес розробки архітектури;
- керівництва і методики проектування для ADM;
- фреймворк архітектурного опису (Architecture Content Framework), що є детально відпрацьованої моделлю результатів розробки;
- архітектурний континуум організації (Enterprise Continuum), у вигляді репозиторію архітектурних артефактів і реалізацій;
- еталонні моделі TOGAF (TOGAF Reference Models): TRM (Technical Reference Model) – технічна еталонна модель; III-RM (The Integrated Information Infrastructure Model) – інтегрована модель інформаційної інфраструктури.
- фреймворк, що описує структуру організації, її персонал, необхідні ролі і рівні відповідальності (Architecture Capability Framework).

Відповідно до методики ADM архітектурний процес можна розбити на дев'ять фаз. ADM представляє з себе ітераційний процес, який відбувається на двох рівнях. На верхньому рівні кожної ітерації повторюються загальні для кожної з фаз дії. Нижній рівень описує ітерації всередині кожної фази. Рішення приймаються на підставі існуючих вимог бізнесу та існуючих рішень.

Схема фаз TOGAF представлена на рис. 2.11.

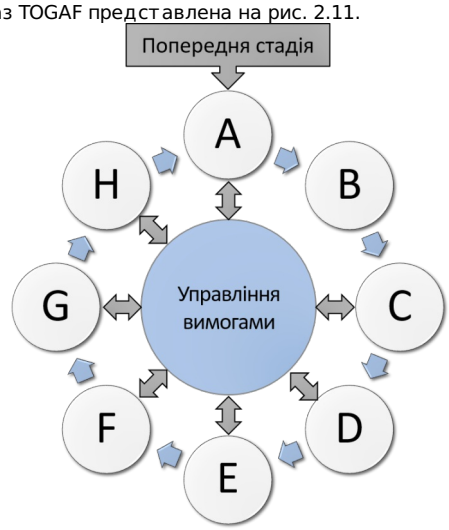


Рисунок 2.11 – Схема фаз відповідно методиці ADM

Таблиця 2.2 – Зміст фаз архітектурного процесу методики ADM

Стадія процесу	Опис
Попередня фаза (Preliminary Phase)	Визначення способів управління, меж розробки, принципів реалізації, уточнення моделі щодо специфіки .
Фаза A, розробка загального подання (Architecture Vision)	Створення загального уявлення про архітектуру, визначення меж проекту, затвердження плану робіт і завдань для виконавців
Фаза B, розробка бізнес-архітектури (Business Architecture)	Виявлення принципів функціонування організації, принципів управління проектом
Фаза C, розробка інформаційної архітектури (Information Systems Architecture)	Розробка архітектури даних і додатків
Фаза D, розробка технологічної архітектури (Technology Architecture)	Підбір апаратних засобів, засобів мережевої інфраструктури, механизмів їх взаємодії
Фаза E, Можливості і рішення (Opportunities and Solutions)	Реалізація розробленої архітектури (купити готове рішення або зробити власне)
Фаза F, Планування переходу до нової архітектури (Migration Planning)	Оцінка ризиків, аналіз деталей переходу
Фаза G, формування системи керування реалізацією (Implementation Governance)	Моніторинг процесу впровадження і специфікація виникаючих проблем
Фаза H, керування зміною архітектури (Architecture Change Management)	Налагодження процесу керування змінами розробленої архітектури

Слід зазначити, що TOGAF може використовуватися не тільки як єдиний фреймворк при розробці, але і в сукупності з іншими фреймворками. Зокрема, до нього входить спеціальний документ, що визначає відповідності між поняттями TOGAF і фреймворком Захмана.

2.9.3. Фреймворк DoDAF.

Фреймворк міністерства оборони США DoDAF (Department of Defense Architecture Framework) складається з трьох основних елементів: моделі (models), види (views) і точки зору (viewpoints). Цей набір дозволяється відображати погляди всіх зацікавлених сторін. Незважаючи на військове призначення, DoDAF є вільно поширюваним. На його базі були сформовані фремворкі НАТО (NATO Architecture Framework – NAF), фреймворк Міністерства Оборони Великобританії (Ministry of Defense Architecture Framework – MODAF) і безліч інших.

Головна особливість DoDAF – орієнтація на дані, що має на увазі підвищену важливість збереження даних і повторного їх використання. Основним класом систем, що проектуються за допомогою цього фреймворку, є системи збору, зберігання та аналізу даних для підтримки прийняття рішень (СППР).

DoDAF визначає моделі, як шаблони для збору даних і поділяє їх на класи:

- таблиці;
- графічні зображення структурних аспектів архітектурного рішення;
- графічні зображення поведінкових аспектів архітектурного рішення;
- відображення, що визначають взаємозв'язок між типами інформації;
- онтології;
- картинки у вільному форматі;
- тимчасові діаграми.

Види визначаються способами подання для користувача пов'язаного набору даних. До видів можна віднести документи, таблиці, графіки, діаграми і т.д.

Точки зору собою впорядковану безліч видів.

Друга версія DoDAF містить вісім точок зору:

- узагальнена (All Viewpoint): інтегрує всі точки зору для створення архітектурного контексту;
- визначальна потенційні можливості (Capability Viewpoint): навчання персоналу, терміни поставок і т.д .;
- визначальна дані та інформацію (Data and Information Viewpoint): визначає способи подання та структури даних;
- операційна (Operational Viewpoint): розглядає сценарії роботи та активності системи;
- проектна (Project Viewpoint): розглядає необхідні характеристики і можливості системи;
- сервісна (Service Viewpoint): розглядає сукупність сервісів;
- враховує стандарти (Standards Viewpoint): розглядає технічні стандарти, обмеження, методики, керівництва і т.д.;
- системна (System Viewpoint): розглядає сукупність взаємодіючих систем та їх взаємодію.

DoDAF використовує мета-модель даних (Data Meta-Model - DM2), яка є онтологією, складеної з рівнів, що відбивають особливості подання інформації для конкретних груп користувачів. DM2 може бути розширена. У ній визначено три рівні:

1. Концептуальна модель даних (Conceptual Data Model) - описує архітектуру в нетехнічні термінах.
2. Логічна модель даних (Logical Data Model) - розширення концептуальної моделі шляхом додавання атрибутів.
3. Специфікація обміну даними на фізичному рівні (Physical Exchange Specification) - засіб, що забезпечує обмін інформацією між моделями.

Існує вісім базових принципів, керуючись якими, можна успішно застосовувати DoDAF [Рад]:

1. Архітектурне опис має бути чітко орієнтоване на проголошені цілі.
2. Архітектурне опис має бути по можливості простим і зрозумілим, але не спрощеним.
3. Архітектурноеопісаніе должнооблегчать, а не ускладнювати процес прийняття рішень.
4. Архітектурне опис має бути складено таким чином, щоб його можна було використовувати для порівняння різних архітектур.
5. При складанні архітектурного опису повинні в максимальному ступені використовуватися стандартні типи даних, що визначаються в DM2.
6. Архітектурне опис має виконуватися в термінах самих даних, а не інструментальних засобів роботи з даними.
7. Архітектурні дані повинні бути організовані у вигляді, зручному для групової роботи.
8. Архітектурне опис має бути побудовано таким чином, щоб його можна було використовувати в мережевому середовищі.

DoDAF призначений для складання архітектурного опису системи. Результатом його застосування буде набір документів, а не інформаційна система.

2.10 ІНТЕГРАЦІЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Термін інтеграція має широке значення. Під ним можна розуміти об'єднання інформаційних систем, додатків, різних компаній або людей.

Інтеграція поділяється на зовнішню і внутрішню. Внутрішня інтеграція передбачає об'єднання корпоративних додатків в одній організації (Enterprise Application Integration), а зовнішня – інтеграцію інформаційних систем організацій (Business-to-Business Application).

2.10.1. Інтеграційні підходи.

Існують чотири основні типові інтеграційні підходи:

- інтеграція на рівні даних;
- інтеграція на рівні бізнес-функцій і бізнес-об'єктів;
- інтеграція на рівні бізнес-процесів;
- портали.

Інтеграція на рівні даних (Information- Oriented Integration) має на увазі наявність в системах баз даних, для роботи з якими необхідно розробити єдиний програмний інтерфейс.

До основних технологічних рішень даного підходу відносяться:

- системи реплікації даних;
- федеративні бази даних;
- використання API для доступу до EPR- системам.

Реплікація є процесом синхронізації даних між різними джерелами. Необхідність у цьому виникає в момент зміни блоку інформації в розподілених системах зберігання для гарантії коректності і несуперечності даних, які використовуються в усіх модулях або додатках інформаційної системи. Зазвичай функції реплікації покладають на проміжне програмне забезпечення.

Федеративні бази даних (Federated Database Systems) надають єдиний інтерфейс до розподілених даних. Це забезпечує інтеграцію безлічі автономних даних, які можуть бути фізично розташовані на різних пристроях в мережі. Такі бази даних прийнято називати віртуальними.

Використання API для доступу до ERP-систем покликане спростити механізми обміну інформацією між одними додатками і програмним забезпеченням, призначеним для управління функціонуванням виробничих інформаційних систем (ERP).

Інтеграція на рівні бізнес-функцій і бізнес об'єктів передбачає реалізацію спільно використовуваних служб (сервісів). Служба може бути набором функцій, використовуваному в декількох додатках. Набір служб і буде бізнес-функціями.

При використанні сервіс-орієнтованої архітектури, бізнес-функції можна розглядати як бізнес-сервіси, а при компонентному підході – бізнес-об'єктами (бізнес-компонентами).

Інтеграція на рівні бізнес-процесів розрізняється залежно від рівня інтеграції. При внутрішній інтеграції взаємодіє велика кількість сервісів, а при зовнішній інтеграції, в основному, два. Самі бізнес-процеси функціонують над виділеними службами, для управління якими існує спеціальний інтерпретована мова.

Портали можна вважати графічними інтерфейсами бізнес-процесів, оскільки вони призначені для персоналізованого доступу до інформації та консолідації даних з декількох джерел.

Головне призначення процесу інтеграції – об'єднання функцій додатків або модулів для надання нової функціональності.

При інтеграції додатків можна виділити два основних типи завдань:

- завдання інтеграції корпоративних додатків;
- завдання інтеграції додатків з різних інформаційних систем.

Для вирішення завдань першого типу застосовуються системи EAI, які іноді називаються A2A (Application-to-Application Integration), а для вирішення завдань другого типу застосовуються системи B2B (Business-to-Business Integration).

У деяких ситуаціях дуже складно визначити різницю між інтеграцією A2A і B2B, оскільки складність деяких рішень всередині інформаційних систем може перевищувати складність рішень для їх спільного функціонування.

2.10.2. Топології інтеграції.

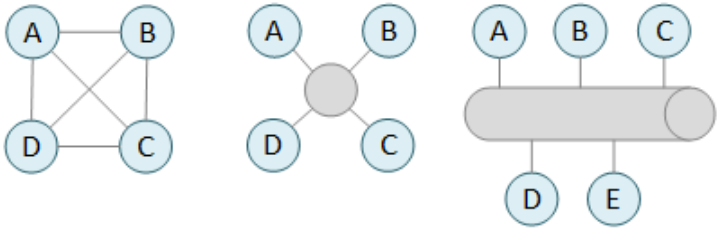
Існують три альтернативних топології інтеграції:

1. Точка-точка (Point-to-Point).
2. Шлюз (hub-and-spoke).
3. Шина (Bus).

У топології «точка-точка» всі об'єкти мають прямі зв'язки один з одним (рис. 2.12, а). Слід зазначити, що кожен зв'язок може бути реалізований яким завгодно способом. Варіанти реалізації залежать від вимог і характеристик взаємодії між об'єктами. До недоліків топології можна віднести:

- недостатня гнучкість;
- складність підтримки численних з'єднань «точка-точка»;
- зміни одного об'єкта впливають на решту;
- логіка маршрутизації часто програмується в коді об'єктів;
- відсутність загальної моделі безпеки;
- використання різних API;
- низька надійність;
- складність створення фреймворків;
- складність підтримки асинхронного взаємодії;

Для скорочення числа використовуваних інтерфейсів слід використовувати топологію із загальним шлюзом (рис. 2.12, б) або топологію із загальною шиною (рис. 2.12, в). Такі моделі інтеграції реалізуються на рівні проміжного програмного забезпечення.



а) б) с)

Рисунок 2.12 – Класифікація топологій інтеграції систем

Наступним кроком у розробці інтеграційних архітектур можна вважати появу корпоративної сервісної шини (Enterprise Service Bus – ESB). Ряд авторів розглядають системи ESB, як наступну сходинку розвитку EAI. Однак є кілька відмінностей:

- EAI – централізована архітектура, з обміном інформації через хаб (брокер), а ESB – шинна архітектура, яка може бути реалізована у вигляді декількох розподілених систем;
- на відміну від EAI, ESB орієнтована на використання відкритих стандартів.

Ці дві відмінності наочно демонструють можливість використання ESB як інтеграційної платформи, що дозволяє використовувати різні механізми інтеграції. ESB дозволяє проводити

як внутрішню, так і зовнішню інтеграції, і являє собою шину (backbone), що працює як слабкозв'язана система, керована подіями.

Концепції сервіс-орієнтованих архітектур (СОА) і ESB дуже сильно пов'язані. ESB підтримує принцип реалізації СОА: поділ подання служби і її реалізації.

Функції ESB:

- надання інтерфейсів взаємодії;
- відправлення і маршрутизація повідомлень;
- перетворення даних;
- реакція на події;
- управління політиками;
- віртуалізація.

На підставі функцій ESB, можна сформувати типовий список вимог, що пред'являються з боку користувачів:

- велика пропускна здатність;
- підтримка декількох стилів інтеграції;
- забезпечення можливості додатків працювати з сервісами як безпосередньо, так і через адаптери.

ESB є, по суті, логічним компонентом архітектури, що призводить інтеграційну інфраструктуру у відповідність принципу СОА. Архітектурами, побудованими за принципом ESB, складніше управляти, але вони більш гнучкі і масштабовані, більш того, впровадження СОА не зажадає змін у всіх елементах системи, в результаті чого зможе відбуватися поетапно.

Можна уявити ESB у вигляді п'ятирівневої структури:

- рівень сполучення (адаптери та інтерфейси);
- транспортна підсистема;
- рівень реалізації бізнес-логіки;
- рівень управління бізнес-процесами;
- рівень бізнес-управління.

Рівень сполучення покликаний вирішувати проблему використання різних інтерфейсів. На цьому рівні функціонують адаптери, які відстежують події в додатках і в інтеграційній підсистемі, і забезпечують перетворення переданих об'єктів при взаємодії з транспортною підсистемою. Існує можливість, окрім заздалегідь створених адаптерів інтеграційної платформи, використовувати створені самостійно.

Адаптери можна розділити на дві категорії: технологічні (застосовуються для інтеграції технологічних компонент, за відсутності у них API), адаптери для додатків (застосовуються для інтеграції з конкретним додатком).

Транспортна підсистема надає можливість асинхронного взаємодії інтегрованим додаткам. Даний рівень також відповідає за управління та безпеку інформації, може виконувати маршрутизацію повідомлень і їх обробку.

Рівень реалізації бізнес-логіки надає функції для трансформації і маршрутизації повідомлень. На цьому рівні функціонують брокери повідомлень, що обмінюються повідомленнями через транспортну підсистему.

Брокер повідомлень може виконувати такі функції:

1. Прийняття повідомлень і їх відправка за вказаними адресами.
2. Перетворення форматів повідомлень.
3. Агрегування і фрагментація повідомлень.
4. Взаємодія з репозиторіями.
5. Вибірка даних через виклики Web-служб.
6. Обробка помилок і подій.
7. Маршрутизація повідомлень за адресою, вмістом, темою.

Управлення бізнес-процесами на однойменному рівні здійснюється за допомогою BPEL (Business Process Execution Language) за допомогою web-сервісів.

Рівень бізнес-управління представляє з себе надбудову на попередньому рівню і призначений для управління бізнес-процесами в термінах відповідної предметної області.

Підхід ESB має безліч переваг і дозволяє будувати інтеграційні архітектури практично будь-якої складності. Типова структура інтеграційної системи представлена на рис. 2.13.

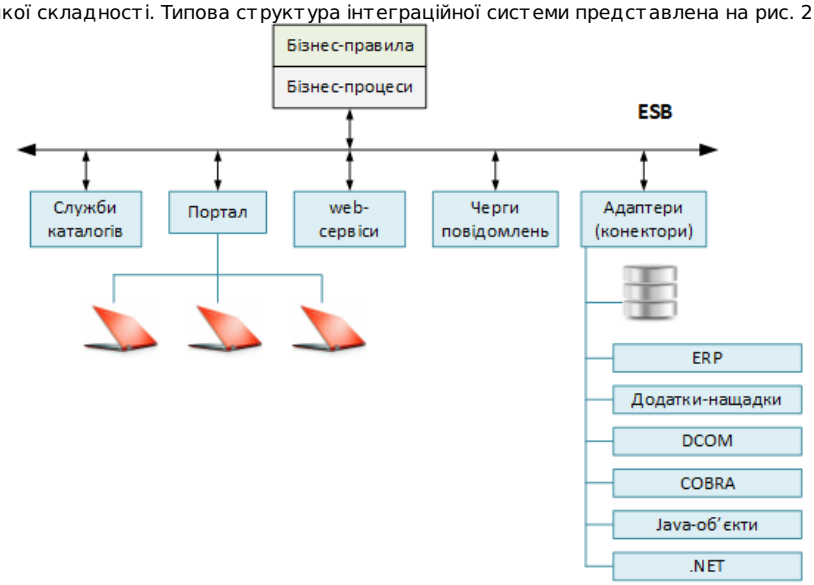


Рисунок 2.13 – Типова структура інтеграційної системи

ВИСНОВКИ

Під архітектурою програмних систем розуміється сукупність рішень щодо:

- організації програмної системи;
- вибору структурних елементів, що складають систему і їх інтерфейсів;
- поведінки цих елементів у взаємодії з іншими елементами;
- об'єднання цих елементів у підсистеми;
- архітектурного стилю, що визначає логічну й фізичну організацію системи: статичні і динамічні елементи, їх інтерфейси і способи їх об'єднання.

Архітектура програмної системи охоплює не тільки її структурні і поведінкові аспекти, але й правила її використання та інтеграції з іншими системами, функціональність, продуктивність, гнучкість, надійність, можливість повторного застосування, повноту, економічні та технологічні обмеження, а також питання для користувача інтерфейсу.

По мірі розвитку програмних систем все більшого значення набуває їх інтеграція одна з одною з метою побудови єдиного інформаційного простору підприємства. Як можна бачити з вищенаведених визначень інтеграція є найважливішим елементом архітектури.

Для того, щоб побудувати правильну і надійну архітектуру і грамотно спроектувати інтеграцію програмних систем необхідно чітко слідувати сучасним стандартам в цих областях. Без цього велика ймовірність створити архітектуру, яка нездатна розвиватися і задовольняти зростаючим потребам користувачів IT. В якості законодавців стандартів у цій галузі виступають такі міжнародні організації як SEI (Software Engineering Institute), WWW (консорціум World Wide Web), OMG (Object Management Group), організація розробників Java - JCP (Java Community Process), IEEE (Institute of Electrical and Electronics Engineers) та інші

Кожен архітектор інформаційних систем повинен враховувати три важливі рекомендації.

1. Контролюйте рамки проекту.
2. Завжди пам'ятайте, для чого потрібна проектуєма модель. Моделі даних призначені для реалізації, а інформаційні моделі – для документації та обміну інформацією з людьми.
3. Не дозволяйте інструментальним засобам визначати ваш погляд на речі. Важко документувати обширні і несуворі відношення спадкування, використовуючи тільки SQL або ER-діаграми.

3.1 МОДЕЛЮВАННЯ І МОДЕЛІ ІНФОРМАЦІЙНИХ СИСТЕМ

3.1.1. Поняття моделі і моделювання.

Модель (лат. "modulus" – міра) – об'єкт-замінник об'єкта-оригіналу, що забезпечує вивчення деяких властивостей останнього; спрощене подання системи для її аналізу і передбачення, а також отримання якісних та кількісних результатів, необхідних для прийняття правильного управлінського рішення.

При вирішенні конкретної задачі, коли необхідно виявити певну властивість досліджуваного об'єкта, модель виявляється не тільки корисним, але й часом єдиним інструментом дослідження. Один і той самий об'єкт може мати безліч моделей, а різні об'єкти можуть описуватися однією моделлю.

Єдина класифікація видів моделей відсутня через багатозначність поняття "модель" в науці і техніці. Її можна проводити за різними підставами: за характером моделей і модельованих об'єктів; за сферами додатків та ін.

Під терміном "моделювання" зазвичай розуміють процес створення точного опису системи; метод пізнання, що складається в створенні і дослідженні моделей.

Моделювання полегшує вивчення об'єкта з метою його створення, подальшого перетворення і розвитку. Воно використовується для дослідження існуючої системи, коли реальний експеримент проводити недоцільно через значні фінансові і трудові витрат, а також при необхідності проведення аналізу проектованої системи, тобто яка ще фізично не існує в даній організації.

Для формування моделі використовуються:

- структурна схема об'єкта;
- структурно-функціональна схема об'єкта;
- алгоритми функціонування системи;
- схема розташування технічних засобів на об'єкті;
- схема зв'язку та ін.

Всі моделі можна розбити на два великі класи: предметні (матеріальні) і знакові (інформаційні).

Для проектування ІС використовують інформаційні моделі, що представляють об'єкти і процеси у формі рисунків, схем, креслень, таблиць, формул, текстів і т.п.

Інформаційна модель – це модель об'єкта, процесу або явища, в якій представлені

інформаційні аспекти модельованого об'єкту, процесу або явища. Вона є основою розробки моделей ІС.

Для створення описових текстових інформаційних моделей зазвичай використовують природні мови. Поряд з природними розроблені і використовуються формальні мови: системи числення, алгебра відношень, мови програмування та ін. Основна відмінність формальних мов від природних полягає в наявність у формальних мов не тільки жорстко зафіксованого алфавіту, але і строгих правил граматики та синтаксису.

За допомогою формальних мов будують інформаційні моделі певного типу – формально-логічні моделі.

При вивченні нового об'єкта спочатку зазвичай будується його описова модель, потім вона формалізується, тобто документується з використанням математичних формул, геометричних об'єктів і т.д.

Процес побудови інформаційних моделей за допомогою формальних мов називають формалізацією.

Моделі, побудовані з використанням математичних понять і формул, називають математичними моделями.

Модель повинна враховувати якомога більше число факторів. Однак реалізувати таке положення складно особливо в слабкоструктурованих системах. Тому найчастіше прагнуть створювати моделі досить простих елементів, з урахуванням їх мікро- і макрозв'язків. Це дозволяє отримувати адекватні результати.

Часткова класифікація методів моделювання представлена на рис. 3.1.

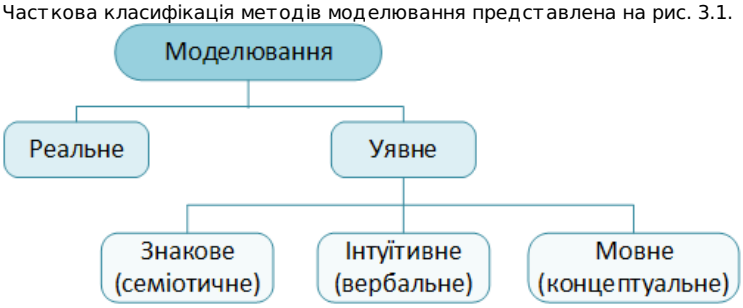


Рисунок 3.1 – Класифікація методів моделювання

Зазвичай розрізняють реальне (матеріальне, предметне) і уявне (ідеалізоване, концептуально-методологічне) моделювання.

Концептуально-методологічне моделювання є процесом встановлення відповідності реальному об'єкту деякої абстрактної конструкції, що дозволяє отримати характеристики об'єкта. Дана модель, як і всяка інша, описує реальний об'єкт лише з деякою ступенем наближення до дійсності.

Концептуальне моделювання є структурованим процесом створення систем, що складається з таких етапів:

1. Аналіз,
2. Проектування,
3. Програмування,
4. Тестування,
5. Впровадження.

Найважливішою формою системного аналізу складних систем є імітаційне моделювання на ЕОМ, що описує процеси функціонування систем у вигляді алгоритмів. Його застосовують у випадках, коли необхідно врахувати велику різноманітність вихідних даних, вивчити протікання процесів в різних умовах. Процес імітації на будь-якому етапі може бути призупинений для проведення наукового експерименту на вербальному (описовому) рівні, результати якого після оцінки та обробки можуть бути використані на наступних етапах імітації.

Існує декількох методів і принципів побудови інформаційних систем (автоматизованих ІС), серед яких можна виділити: методи "знизу-вверх" і "зверху-вниз", принципи "дуалізму", багатокомпонентності та ін.

3.1.2. Метод "знизу-догори".

Досвід і методи роботи вітчизняних програмістів сформувалися у великих обчислювальних центрах (ВЦ), основною метою яких було не створення тиражованих продуктів, а виконання завдань конкретної установи. Сучасні керівники часто вдаються до нього, вважаючи, що їм зручно мати своїх фахівців. Розробка програм "знизу-догори", здійснювана кваліфікованими програмістами, дозволяє автоматизувати, як правило, окремі робочі процеси. Такий метод досить витратний і все рідше використовується, особливо в малих і середніх підприємствах.

3.1.3. Метод "згори-донизу".

Розвиток комерційних та інших сучасних структур послужив підставою до формування ринку різних програмних засобів. Найбільший розвиток одержали ІС, орієнтовані на автоматизацію ведення бухгалтерського аналітичного обліку і технологічних процесів. В результаті з'явилися ІС, розроблені сторонніми, як правило, спеціалізованими організаціями і групами фахівців "згори", в припущенні, що одна ІС зможе задовольняти потреби багатьох користувачів.

Така ідея обмежила можливості розробників в структурі інформаційних множин БД, у використанні варіантів екранних форм, алгоритмів розрахунку і, отже, позбавила можливості принципово розширити коло вирішуваних завдань. Закладені "зверху" жорсткі рамки ("загальні для всіх") обмежують можливості ІС. Стало очевидним, що для успішної реалізації завдань повної автоматизації організації слід змінювати ідеологію побудови автоматизованих інформаційних систем (АІС).

3.1.4. Принципи "дуалізму" і багатокомпонентності.

Розвиток систем і підприємств, збільшення числа їх філій та клієнтів, підвищення якості обслуговування та інше викликали істотні зміни в розробці та функціонуванні АІС, в основному базуються на збалансованому поєднанні двох попередніх методів.

Новий підхід орієнтований на спеціалізоване програмне забезпечення (СПЗ), можливість адаптації програмного апарату до практично будь-яких умов і різним вимогам інструктивних матеріалів і прийнятим правилам роботи. Крім того, гнучка система налаштувань СПЗ в АІС при проведенні модернізації одного з компонентів дозволяє не зачіпати центральну частину (ядро) АІС і інші її компоненти, що значно підвищує надійність, тривалість життя ІС і забезпечує найбільш повне виконання необхідних функцій.

Такий підхід ліг в основу "принципу дуалізму". Його реалізація зажадала побудови АІС нового покоління у вигляді програмних модулів, органічно пов'язаних між собою, але в той же час здатних працювати автономно.

Багатокомпонентна система забезпечує дотримання основоположного принципу побудови АІС – відсутності дублювання введення вихідних даних.

Інформація за операціями, проведеними із застосуванням одного з компонентів системи, може використовуватися будь-яким іншим її компонентом. Модульність побудови АІС нового покоління і принцип одноразового введення дають можливість гнучко варіювати конфігурацією цих систем. Така структура дозволяє включити в АІС нового покоління компоненти створення сховищ даних, розділяючи системи оперативної дії та системи підтримки прийняття рішення.

Крім того, однією з переваг принципу багатокомпонентності, що є базовим при створенні АІС нового покоління, є можливість поетапного впровадження ІС. На першому етапі впровадження встановлюють або замінюють вже застарілі компоненти ІС, що потребують оновлення ПЗ. На другому етапі відбувається розвиток системи з під'єднанням нових компонентів і відпрацюванням межкомпонентних зв'язків. Можливість застосування такої методики впровадження забезпечує її досить просте тиражування та адаптацію до місцевих умов.

Зі сказаного можна припустити, що автоматизована інформаційна система нового покоління – це багатокомпонентна система з розподіленою базою даних.

3.1.5. Використання моделей при створенні ІС.

Процеси створення моделей носять етапний характер. Основні види моделей, типу "каскад" ("водоспад"), "ітеративність" і "спіраль" описані у першій лекції. Повернення до їх розгляду пов'язано з особливостями використання цих моделей у процесі розробки ІС.

Каскадна модель ІС.

Каскадна модель ІС складається з послідовно виконуваних етапів. Кожен етап повністю закінчується до того, як почнеться наступний. Етапи не перекриваються у часі: наступний етап не починається доти, доки не завершиться попередній. Повернення до попередніх етапів не передбачений або всіляко обмежений. Виправлення помилок відбувається лише на стадії тестування. Результат з'являється тільки в кінці розробки ІС. Критерієм появи результату є відсутність помилок і точну відповідність отриманої ІС первісної її специфікації.

Для цієї моделі характерна автоматизація окремих незв'язаних задач, що не вимагає виконання інформаційної інтеграції та сумісності, програмного, технічного та організаційного сполучення. В рамках вирішення окремих завдань каскадна модель за термінами розробки та надійності виправдовувала себе. Застосування каскадної моделі до великих і складних проектів внаслідок великої тривалості процесу проектування та мінливості вимог за цей час призводить до їх практичної нереалізуємості.

Поетапна (ітераційна) модель з проміжним контролем.

Використовується послідовність розташування етапів створення ІС. Але кожен наступний етап має зворотний зв'язок з попередніми етапами. Виправлення помилок відбувається на кожному з етапів, відразу при виявленні проблеми – проміжний контроль. Наступний етап не починається, поки не завершиться попередній. При першому проході по моделі згори донизу, як тільки виявлена помилка, здійснюється повернення до попередніх етапів (знизу догори), що викликав помилку. Етапи виявляються розтягнутими в часі. Результат з'являється тільки в кінці розробки ІС, як і в моделі "водоспад".

Спіральна модель.

У цій моделі результат з'являється фактично на кожному витку спіралі. Цей проміжний результат аналізується, та виявлені недоліки ІС спонукають проведення наступного витка спіралі. Таким чином послідовно конкретизуються деталі проекту і в підсумку вибирається і доводиться до реалізації обґрунтований варіант. Спіраль завершується тоді, коли клієнт і розробник приходять до згоди щодо отриманого результату.

Модель складається з послідовно розташованих етапів (як і "водоспад") в межах одного витка спіралі. Усередині витка спіралі етапи не мають зворотного зв'язку. Аналіз результату здійснюється в кінці витка і ініціює новий виток спіралі. Виправлення помилок відбувається при тестуванні на кожному витку спіралі. Помилки, які не можуть бути виправлені і вимагають більш глибоких структурних змін, ініціюють новий виток спіралі. Етапи можуть перекриватися в часі в межах одного витка спіралі. Результат з'являється в кінці кожного витка спіралі і піддається детальному аналізу. При переході від витка до витка відбувається накопичення і повторне використання програмних засобів, моделей і прототипів. Процес орієнтований на розвиток і модифікацію ІС в процесі її проектування, на аналіз ризиків і витрат під час проектування.

Основна особливість даного методу полягає в концентрації складності на початкових етапах розробки ІС (аналіз, проектування). Складність і трудомісткість наступних етапів в межах одного витка спіралі відносно невисокі. При цьому методі пропонується спосіб зниження витрат у цілому при розробці ІС (і будь-якого іншого ПЗ) за рахунок запобігання потенційних помилок на етапах її аналізу і проектування. При цьому використовується підхід до організації проектування ІС "зверху-вниз", коли спочатку визначається склад функціональних підсистем, а потім постановка окремих завдань.

3.1.6. Автоматизована система моделювання.

Процеси моделювання все частіше здійснюються з використанням спеціальних комп'ютерних програмних засобів, що дозволяють автоматизувати цю діяльність.

Автоматизована система моделювання (АСМ) – комп'ютерна система, призначена для надання допомоги користувачеві за поданням потрібної йому задачі у вигляді певної математичної схеми, прийнятої в даній системі, вирішити задачу (провести моделювання з отриманої схеми) і проаналізувати результати.

АСМ складається з трьох основних компонентів: функціональне наповнення, мова завдань і системне наповнення.

Функціональне наповнення є сукупністю конструктивних елементів (модулів), з яких складається схема (модель).

Системне наповнення – це набір програм, що відбивають специфіку реалізації АСМ і забезпечують власне функціонування системи: трансляцію і виконання завдань, підтримку бази знань про предметну область і т.д.

Мова завдань (МЗ) служить для опису завдань, що вводяться в систему.

Засобами та інструментом автоматизованого проектування та розробки інформаційних систем є CASE-засоби і системи, орієнтовані на підтримку розробки інформаційних систем.

3.2 КЛАСИФІКАЦІЯ МОДЕЛЕЙ ІНФОРМАЦІЙНИХ СИСТЕМ

При аналізі і особливо при проектуванні системи повинні бути побудовані її повні і несуперечливі моделі. При цьому під моделлю розуміється сукупність взаємопов'язаних абстрактних елементів з можливою вказівкою їх властивостей, поведінки та зв'язків між ними.

Класифікувати моделі можна за такими ознаками.

1. За строгістю опису:

Неформальні – представлені в неструктурованому вигляді і дають загальне уявлення про системи, які моделюються. Недостатньо наочні (особливо при складній взаємодії між об'єктами) і неприйнятні для будь-якого кількісного аналізу та обробки автоматичними засобами;

Формальні:

- описові – моделі, де відомості представлені за допомогою спеціальних документів (бланки, форми, анкети, таблиці і т. п.);
- графічні – моделі являють собою схеми, креслення, графи, діаграми і т. д. Найбільш наочні і набули широкого поширення при проектуванні за допомогою CASE-засобів;
- математичні – представляють модель мовою математичних відносин у вигляді функціональних залежностей, систем алгебраїчних або диференціальних рівнянь, логічних виразів і т. д.

2. За ступенем фізичної реалізації (логічної незалежності):

Логічні – описують склад, структуру, стан або поведінку елементів системи без прив'язки до конкретних мов або середовищ програмування, СУБД, технічних засобів і т. д. При розробці системи це забезпечує гнучкість у виборі і швидкий перехід з однієї програмно-апаратної платформи на іншу;

Фізичні – описують елементи системи відповідно до прийнятої фізичної реалізації цих елементів (мов програмування, СУБД, пристроїв і т. д.);

3. За ступенем відображення динаміки процесів:

Статичні – описують склад і структуру системи.

Динамічні – описують поведінку системи та / або її окремих елементів. Як правило, такі моделі описують порядок дій або стан системи і переходи між ними. Іншими словами, в цих моделях явно чи не явно присутнє поняття часу;

4. За відображаємим аспектом:

Функціональні – описують функції системи, можливі варіанти її використання; можуть містити відомості про циркулюючу в системі інформацію, об'єкти та суб'єкти, що взаємодіють з системою; можуть бути як динамічними, так і статичними моделями;

Інформаційні – описують склад і структуру даних (реляційних БД, класів та ін.). Відносяться до статичних моделей;

Поведінкові – описують стану системи та/ або її окремих елементів і переходи між ними, взаємодію елементів, алгоритми обробки інформації. Відносяться до динамічних моделей;

Компонентні – описують склад і структуру програмних і апаратних засобів. Відносяться до статичних моделей;

Змішані – характеризують відразу кілька аспектів системи (наприклад, діаграми потоків даних відображують роботи, накопичувачі даних, підсистеми) і т. д.

На стадіях формування та аналізу вимоги спочатку починають з побудови неформальних моделей (змістовного опису предметної області), поступово переходячи до формальних. Аналогічно на стадії проектування починають зі створення формальних логічних моделей і закінчують фізичними. Одним з найважливіших результатів проектування є набір логічних і фізичних моделей, що описують всі аспекти системи. Цей набір повинен бути достатнім для подальшої реалізації системи на стадії кодування.

3.3 ІНФОРМАЦІЙНА (КОНЦЕПТУАЛЬНА) МОДЕЛЬ ІС

Етапу аналізу передує етап визначення вимог до ІС, в процесі якого виявляються проблеми предметної області (виконується проблематизація предметної області) і формуються пропозиції щодо розв'язання виявлених проблем за допомогою ІС.

Пропозиції щодо вирішення проблем відносяться, насамперед, до функціональних можливостей, якими повинна володіти інформаційна система, щоб успішно вирішувати виявлені проблеми. Після формування списку функціональних можливостей (у контексті концепції – функціональних вимог) і основних понять, якими оперуватимуть функціональні можливості, переходять до виконання робіт етапу аналізу, а потім і проектування. Результати етапу визначення вимог є вхідними даними етапу аналізу.

Традиційно в курсах лекцій і технічних статтях за методологією RUP, функціональні вимоги до інформаційної системи оформлюються у вигляді діаграми прецедентів (use case). За допомогою діаграм прецедентів (а детальніше і діаграм активності або станів) моделюється предметна область, причому в аспектах комплексної архітектури (схеми Захмана). Тому вхідними даними для аналізу служить перелік функціональних можливостей і основних понять з розділу концепції

Під аналізом будемо розуміти процес аналізу вимог (і перш все, функціональних вимог) до інформаційної системи, сформованих на етапі визначення вимог. У процесі аналізу вирішується завдання аналізу, результати якого: декомпозиція системи на більш дрібні елементи і визначення властивостей системи або середовища, що оточує систему.

Метою аналізу може бути визначення закону перетворення інформації, що задає поведінку системи (наприклад, перетворення функціональних вимог у поведінку системи). При цьому система постає як один елемент, на вхід якого надходять вхідні дані (повідомлення), а на виході, залежно від закону поведінки, формуються вихідні дані, що задовольняють користувача.

Наприклад, в контексті методології RUP результат аналізу це концептуальна схема (діаграма класів, що моделює основні поняття і вирішує задачу декомпозиції) і реалізація варіанта використання у вигляді діаграми взаємодії, моделюючі поведінку системи за допомогою взаємодії об'єктів аналізу. За умови розгляду системи у вигляді одного елемента вхідними даними будуть повідомлення користувача, а вихідними – повідомлення системи про задоволення (або відмову) виконання запитів користувача. Вихідні повідомлення системи в такому випадку називають списком відповідальності системи. Система нібито відповідає за ці повідомлення (в наслідок ці повідомлення можуть бути інтерпретовані як екранні форми документів, рисунки тощо)

Концептуальна (аналітична) модель системи призначена:

1. Подання більш детальної специфікації вимог до системи (більш точну, ніж у специфікаціях на етапі визначення вимог і розробки варіантів використання) і може розглядатися як перший крок до моделі проектування, тобто служити вхідними даними для проектування системи.
2. Подання системи з використанням мови розробників, що дозволяє вводити більше формалізму і може використовуватися для аналізу внутрішніх механізмів системи.
3. Рішення завдання аналізу, яке полягає в декомпозиції системи на більш дрібні елементи і визначення властивостей системи або середовища, що оточує систему. Так, згідно з рекомендаціями RUP, варіанти використання (якщо вони розроблялися) в моделі аналізу перетворюються в діаграми класів аналізу (при необхідності) і діаграми взаємодіючих об'єктів аналізу. Основним елементом декомпозиції в них є клас. Подання варіантів використання засобом діаграм взаємодіючих об'єктів класу аналізу називається «аналізом реалізації варіанту використання» (варіант поведінки системи). Таке подання містить опис реалізації варіанта використання як в термінах взаємодіючих об'єктів класів аналізу, так і в термінах природної мови.

Узагальнена модель аналізу представлена на рис. 3.2.

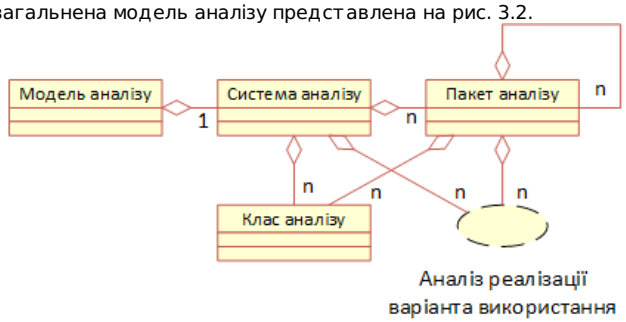


Рисунок 3.2 –Узагальнена модель аналізу

Модель аналізу представлена системою аналізу, яка, у свою чергу, може бути представлена підсистемами (пакетами аналізу). Таке представлення зручно з точки зору управління процесом аналізу та управління проектом в цілому. За допомогою пакетів аналізу можуть бути представлені не тільки абстракції підсистем, а й абстракції рівнів.

Базовими елементами (артефактами) моделі аналізу є класи аналізу.

На основі класів аналізу, як вже зазначалося вище, утворені реалізації варіантів використання, підсистеми, і система в цілому

Аналіз реалізації варіанту використання – організованість (кооперація) всередині концептуальної (аналітичної) моделі, що описує реалізацію і виконання варіанта використання в поняттях класів аналізу і взаємодії об'єктів аналізу.

Розглянемо детальніше поняття класу аналізу.

Клас аналізу являє собою абстракцію одного або декількох класів (і/або підсистем) в проекті системи. Ця абстракція має такі характеристики:

1. Клас аналізу зосереджений на представленні функціональних вимог і відкладає не функціональні вимоги на наступні стадії – проектування і реалізацію. Таке положення визначає клас аналізу як концептуальний клас.
2. Клас аналізу рідко підтримує будь-які інтерфейси в поняттях операцій. Замість цього його поведінка визначена відповідно до відповідальності (обов'язків) верхнього менш формального рівня. Відповідальність в контексті концептуальної моделі це текстовий опис поведінки для розглянутого класу. Клас аналізу визначає атрибути високого рівня
3. На безлічі класів аналізу задаються відношення, які також як і класи аналізу відносять до концептуального рівня подання. Наприклад, спрямованість відношення не надто важлива в аналізі, але суттєва для проектування.
4. Клас аналізу завжди можна віднести до одного з типів: граничний, управляючий, сутності. Кожен з типів має на увазі певну семантику (смислове навантаження), яке призводить до потужного і логічного методу виявлення і описи класів аналізу і сприяє створенню якісної моделі об'єкта.
5. Класи аналізу і задані на них відношення, утворюють концептуальні схеми (необхідні висловлювання). Концептуальні схеми, представлені за допомогою CASE-засобів на якій-небудь формальній мові, (наприклад, на мові UML), будемо називати концептуальними моделями.
6. Класи аналізу для концептуальної схеми (ГОСТ Р 34.320-96) визначаються на підставі таких принципів:

- описів класів (типів) сутностей проблемної області, а не окремих примірників;
- опису понять, менш схильних до змін;
- включення правил чи обмежень, що мають широке вплив на поведінку предметної області (і тому на поведінку концептуальної схеми та інформаційної бази).

У кожному разі повинні дотримуватися загальні принципи концептуальної схеми:

- Принцип 100%, згідно з яким всі загальні аспекти, тобто всі правила, закони і т. д., проблемної області повинні бути описані в концептуальній схемі, причому інформаційна система не може нести відповідальність за недотримання правил і законів, описаних не в концептуальній схемі.
- Принцип концептуалізації, за яким концептуальна схема повинна включати статичні та

динамічні аспекти проблемної області тільки концептуального рівня, не торкаючись зовнішніх і внутрішніх аспектів представлення та організації даних (фізичної організації даних і доступу до них, аспектів подання, що стосуються окремих користувачів).

3.4 ЛОГІЧНА МОДЕЛЬ (МОДЕЛЬ ПРОЕКТУВАННЯ) ІС

Модель проектування – це об'єктна модель, яка описує процес проектування (з RUP конструювання) системи і використовується в якості вихідних даних для процесу реалізації системи. Узагальнена модель проектування представлена на рис. 3.3.

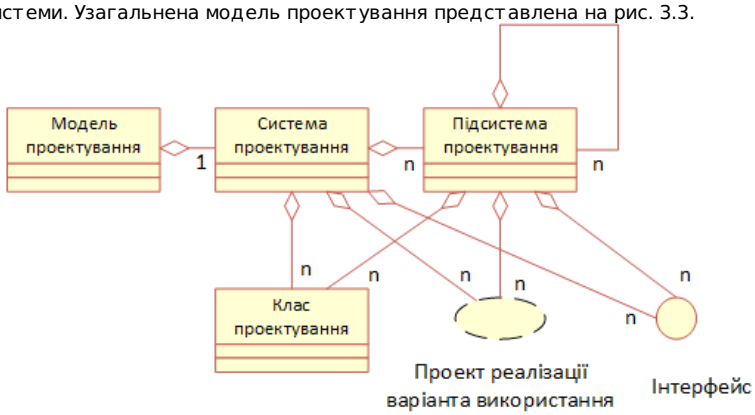


Рисунок 3.3 – Узагальнена модель проектування

Модель проектування представлена системою проектування, яка, в свою чергу, може бути представлена підсистемами (пакетами проектування). Таке подання зручно з точки зору управління процесом проектування та управління проектом в цілому. За допомогою пакетів проектування можуть бути представлені не тільки абстракції підсистем, а й абстракції рівнів.

Розглянемо складові частини логічної моделі проектування.

Базовими елементами (артефактами) моделі проектування є класи проектування. На основі класів проектування утворені проекти реалізації варіантів використання, інтерфейси, підсистеми, і система в цілому. Клас проектування найбільш наближена до реальності абстракція з наступних причин:

1. Мова для опису класу проектування та ж, що і мова програмування для реалізації. Відповідно, операції, параметри, атрибути, типи та інші подробиці визначаються з використанням синтаксису вибраної мови програмування.
2. Зазвичай задається видимість атрибутів і операцій класу проектування (public, protected, private).
3. Відношення, в яких бере участь клас проектування, зазвичай отримують явний вираз при реалізації цього класу. Наприклад, узагальнення має семантику, яка відповідає узагальненню (або спадкування) в мові програмування. Таким чином, узагальнення та агрегація часто відображаються на відповідні змінні (атрибути) реалізації, відповідні посиланнях на об'єкти.
4. Методи (реалізації операцій засобами мови програмування) класу проектування прямо відображаються на відповідні методи класів реалізації (тексти програм)
5. Клас проектування може перекласти обробку деяких вимог на подальшу реалізацію, передавши їй вимоги до реалізації класу. У результаті з'являється можливість відкласти ухвалення рішень, які неможливо прийняти на основі моделі проектування, наприклад, що стосуються питань кодування класу.
6. Клас проектування часто задається стереотипом, який напряму відображається в конструкцію відповідної мови програмування (так для Visual Basic наступні стереотипи – «модуль класу», «форма», «Елемент управління» і т.д.).
7. Клас проектування може бути реалізований у вигляді інтерфейсу, якщо це поняття існує в обраній мові програмування. Наприклад, клас проектування, що представляє клас мови Java може бути реалізований у вигляді інтерфейсу.
8. Клас проектування може бути активним. Це означає, що об'єкти класу будуть використовувати свою власну нитку управління, працюючи паралельно з іншими активними об'єктами. Однак, зазвичай, класи проектування не активні.

Проект реалізації варіанту використання – організованість (кооперація) всередині моделі проектування, що описує реалізацію і виконання варіанта використання в поняттях класів проектування та їх взаємодіючих об'єктів проектування.

Проект реалізації варіантів використання містить текстовий опис потоку подій, діаграми класів і діаграму взаємодій, які відображають потік подій конкретного сценарію варіанта використання, але засобом взаємодій між об'єктами проектування.

Модель поведінки – система взаємодіючих через повідомлення об'єктів, які після їх (повідомлень) визначення повинні бути представлені в моделі класів в якості операцій взаємодії.

Інтерфейси призначені для завдання операцій, які виконуються класом проектування або підсистемою.

Інтерфейси надають спосіб відділення специфікації функціональності в термінах операцій від її реалізації в термінах методів. Це відділення робить клієнтів, які залежать від інтерфейсу або використовують його, незалежними від реалізації інтерфейсу. Окрема реалізація інтерфейсу (клас проектування або підсистема) може бути замінена іншою реалізацією, при цьому ніяких змін в клієнті робити не потрібно.

Більшість інтерфейсів між підсистемами вважається архітектурно значущими, оскільки вони визначають способи взаємодії підсистем. У деяких випадках вони також використовуються для створення стабільних інтерфейсів на початку життєвого циклу системи (ще до того як в підсистемі буде реалізована оголошена функціональність)

3.5 ФУНКЦІОНАЛЬНА МОДЕЛЬ ІС

Третім ключовим моментом створення ІС з метою автоматизації інформаційних процесів організації є аналіз функціональної взаємодії об'єктів автоматизації. Аналітики наводять

результати у вигляді функціональної моделі ПО БД. Склад функціональної моделі істотно залежить від контексту конкретного IT-проекту і може бути представлений за допомогою досить широкого спектра документів у вигляді текстової і графічної інформації.

Визначимо функціональну модель ПО БД як сукупність деяких моделей, призначених для опису процесів обробки інформації. Будемо називати ці моделі конструкціями функціональної моделі. Нижче наведений перелік основних конструкцій функціональної моделі, які необхідні для виконання проектування ІС.

Моделі процесів:

- бізнес-модель процесів (ієрархія функцій системи);
- модель потоку даних.

Моделі станів:

- модель життєвого циклу сутності;
- набір специфікацій функцій системи (вимоги);
- опис функцій системи через сутності й атрибути;
- бізнесу-правила, які реалізують функції.

3.5.1. Бізнес-модель процесів.

Бізнес-модель процесів призначена для опису процесів і функцій системи в ПО ІС. Частіш за все, бізнес-модель документується відповідно нотаціям IDEF0 і подається у вигляді сукупності ієрархічно впорядкованих та взаємопов'язаних діаграм. Діаграми містять такі компоненти:

- контекстна діаграма;
- діаграма декомпозиції;
- діаграма дерева вузлів;
- діаграма «тільки для експозиції».

Контекстна діаграма є вершиною ієрархічної структури діаграм і є узагальненим описом системи та її взаємодії з зовнішнім середовищем.

Подальший опис системи будується послідовним розбиттям функціональної системи на більш детальні фрагменти – діаграми декомпозиції.

Діаграма дерева вузлів передає ієрархічну структуру функцій без відображення взаємозв'язку між ними.

Основними елементами IDEF0-діаграм є роботи, вхідні ти вихідні параметри, керування, механізми та виклик.

3.5.2. Модель потоку даних.

Модель потоку даних призначена для опису процесів переміщення даних в ПО ІС і подається у вигляді діаграми потоку даних (Data Flow Diagram). Основними елементами діаграми є:

- джерела даних (Data Source);
- процеси обробки даних (Data Process);
- сховища даних (Data Store);
- потоки даних (Data Flow).

Джерела даних вказують, хто користується або працює с даними. Процеси обробки вказують на операції, що відбуваються над даними. Сховища даних вказують на місця збереження даних. Потоки даних вказують на спосіб передачі даних між джерелами та сховищами даних.

Для подання діаграм потоку даних частіш за все використовують мережеві структури, які дозволяють дублювання сутностей та відсутність циклів. Потік зображується зліва направо. На діаграмах помічають допустимі та недопустимі напрямки переміщення даних без зазначення процесів керування потоком.

Діаграма потоку даних дозволяє:

- подати систему з точки зору джерел та користувачів даних;
- відобразити переміщення даних в процесі обробки;
- відобразити зовнішні механізми передачі даних;
- відобразити метод отримання даних.
- сховища даних, що дозволить на наступних етапах проектування обґрунтовано визначити кількість БД для ІС;
- прийняті схеми перетворення інформації, що дозволить в подальшому скласти специфікації додатків.

3.5.3. Модель життєвого циклу.

Модель ЖЦ ІС призначена для опису зміни станів ІС та переходів між ними. Модель ЖЦ може бути подана у текстовому вигляді опису.

3.5.4. Набір специфікацій функцій системи.

Набір специфікацій функцій системи (вимог), опис функцій через сутності і атрибути, бізнес-правила.

Аналітики повинні подати проектувальникам набір специфікацій функцій – опис функціональність системи в формі чітко сформульованих бізнес-категорій, згрупованих за напрямками діяльності організації. Іноді подається перелік залежностей між функціями та подіями, що їх викликають.

Текстовий опис функції повинен містити виокремленні сутності та атрибути, а також однозначно інтерпретуватись при читанні.

Після отримання набору специфікацій функцій та опису проектувальник БД може почати розроблення специфікацій модулів додатків БД.

Бізнес-правила подають конкретні вимоги та умови для функцій, які визначають поведінку даних, і використовуються для підтримки цілісності даних в ІС.

ВИСНОВКИ

Якісний аналіз і проектування є необхідними умовами успішного розроблення нетипових інформаційних систем. Саме на початкових стадіях визначаються фундаментальні аспекти моделювання системи і рішення, від яких значною мірою залежить успіх проекту.

Зазначена послідовність розроблення моделей є каркасом, який може бути нарощений залежно від типу та призначення інформаційної системи.

Етап визначення функціональних вимог може супроводжуватись безліччю проблем через складність чітко висловити завдання, які покладаються на ІС, різноманітність поглядів на роботу майбутньої системи, відсутність у замовника знань про можливості сучасних обчислювальних систем та помилкове уявлення про процес автоматизації. Побудова функціональної моделі повинно вирішити велику частину цих проблем.

Наступним етапом є проектування інформаційної моделі, яка має відобразити аспекти організації даних в системі. Цей процес пов'язаний з розробленням моделі бази даних і виконується послідовно такими етапами: концептуальне моделювання, логічне моделювання і фізичне моделювання. Кожен з етапів має результат у вигляді сформованої моделі. Результати виконання попереднього етапу є вхідними для виконання наступного.

В даний час для проектування БД активно використовуються CASE-засоби, в основному орієнтовані на використання ERD (Entity - Relationship Diagrams, діаграми «сутність-зв'язок»). З їх допомогою визначаються важливі для предметної області об'єкти (сутності), відношення один з одним (зв'язки) та їх властивості (атрибути). Слід зазначити, що засоби проектування ERD в основному орієнтовані на реляційні бази даних (РБД), і якщо існує необхідність проектування іншої системи, скажімо об'єктно-орієнтованої, то краще обрати інші методи проектування, як то засобами UML.

Послідовне і якісне розроблення кожної з моделей є чинником успішного завершення проектування ІС.

Підходи до аналізу і проектування інформаційних систем

4.1 CASE-ТЕХНОЛОГІЇ АНАЛІЗУ ТА ПРОЕКТУВАННЯ

CASE-технологія являє собою методологію проектування інформаційних систем, набір методів, нотацій та інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати модель системи на всіх етапах розробки та супроводу системи і розробляти додатки відповідно до інформаційних потреб користувачів.

Під нотацією будемо розуміти встановлені способи відображення елементів системи, тобто графи, таблиці, блок-схеми, формальні і природні мови.

Парадігма – вихідна концептуальна схема (модель) постановки проблеми та її рішення.

Моніторінг – комплексна система спостереження, контролю, оцінки і прогнозу явищ, процесів, об’єктів і т. п.

Як інструментарій реалізації технології використовуються CASE-засоби, основними функціями яких є:

1. Централізоване зберігання в єдиній базі даних проекту (репозитарії) інформації про інформаційну систему протягом всього життєвого циклу. Репозитарій може зберігати об’єкти різних типів: діаграми, визначення екранів і меню, проекти звітів, опис даних, логіку їх обробки, вихідні коди програм і т.п.
2. Пряме проектування програмного забезпечення та баз даних. При цьому порядок використання розробниками CASE-засоби такий:

- створюється логічна модель системи;
- вибирається конкретна мова програмування або СУБД для побудови фізичної моделі, після чого CASE-засіб автоматично створює фізичну модель системи;
- допрацьовується фізична модель;
- виконується автоматична генерація тексту програми або структури бази даних на диску;

3. Зворотне проектування (реінжиніринг). У цьому випадку порядок використання CASE-засоби зворотний – від тексту програми або бази даних на диску до логічної моделі. Крім побудови, CASE-засоби дозволяють швидко інтегрувати отримані таким чином моделі в проект, а також з меншими втратами переходити від однієї фізичної реалізації до іншої.
4. Синхронізація моделей системи з її фізичною реалізацією. У разі зміни моделі системи можуть бути автоматично внесені необхідні зміни в фізичну реалізацію або навпаки.
5. Автоматичне забезпечення якості і тестування моделей на наявність помилок (наприклад, помилок нормалізації БД), повноту і несуперечність.
6. Автоматична генерація документації. Вся документація по проекту генерується автоматично на базі репозитарія (як правило, відповідно до вимог діючих стандартів).

Безсумнівна перевага CASE-технології полягає в тому, що документація завжди відповідає поточному стану справ, оскільки будь-які зміни в проекті автоматично відображаються в репозитарії.

Основна мета використання CASE-технологій полягає в максимальній автоматизації стадій аналізу і проектування систем з метою побудови формальних і несуперечливих моделей системи.

Інша, не менш важлива, мета використання CASE-технологій – винесення частині діяльності зі стадії кодування в стадію проектування.

Більшість сучасних CASE-засобів підтримує методології структурного та / або об'єктно-орієнтованого аналізу і проектування інформаційних систем. Вибір того чи іншого підходу (парадігми) має на увазі дотримання його і на стадії кодування (згідно з принципом концептуальної спільності). Їх відмінність один від одного полягає у виборі способу декомпозиції системи (завдання). Якщо за основу приймається функціональна (алгоритмічна) декомпозиція, то мова йде про структурний підхід, якщо об'єктна – про об'єктно-орієнтований.

Вибір того чи іншого підходу залежить від специфіки розв'язуваної задачі. Як правило, структурний підхід застосовується для автоматизації завдань, що оперують великими обсягами "пасивних" даних і орієнтованих на використання реляційних баз даних (наприклад, облік, збір статистики, математичні та інженерні розрахунки, аналіз даних). Об'єктно-орієнтований підхід в основному орієнтований на вирішення завдань, в яких чітко простежується поділ системи на взаємодіючі між собою сутності (наприклад, імітаційне моделювання, управління технічними об'єктами або технологічними процесами, моніторинг). Найбільш характерна ця особливість для розподілених систем.

Сучасні засоби програмування і управління БД в переважній більшості забезпечують можливість як структурного (процедурного, функціонального), так і об'єктно-орієнтованого програмування. Від розробників залежить, як використовувати ці можливості. Одне можна точно стверджувати, що при побудові інтерфейсу систем остаточно "переміг" об'єктно-орієнтований підхід. Його вже давно не програмує, а "малюють" за допомогою засобів візуальної розробки. При цьому кожен елемент інтерфейсу (поле введення, командна кнопка, перемикачі, таблиці і т. д.) є об'єктом з властивостями, методами та подіями. При програмуванні бізнес-логіки, зберігання і обробки великих обсягів даних методи і засоби структурного підходу ще довго будуть знаходити своє застосування.

4.2 СУТНІСТЬ СТРУКТУРНОГО АНАЛІЗУ І ПРОЕКТУВАННЯ

Протягом еволюції мов та принципів програмування програма стала являти собою не просто єдине ціле з важко помітною внутрішньою структурою, а структуру, що складається з чітко виражених модулів, пов'язаних між собою певними відношеннями (інтерфейсами). Тобто програма набула структури ієрархічної багаторівневої модульної системи. Кожен рівень такої системи є закінченим модулем, підтримуваним і контрольованим модулем, що знаходяться над ним.

Методології структурного аналізу і проектування інформаційних систем з'явилися пізніше фактичного використання цих принципів на практиці (структурного програмування).

При аналізі та проектуванні структурним підходом прийнято називати метод дослідження системи, заснований на представленні її у вигляді ієрархії взаємозалежних функцій. Зазвичай опис системи починається з її загального огляду і потім деталізується, набуваючи ієрархічну структуру з усе більшим числом рівнів. Розбиття на рівні абстракції проводиться з обмеженням числа елементів на кожному з них. Опис кожного рівня включає в себе тільки істотні для цього рівня елементи (принцип абстрагування). Процес розбиття триває аж до конкретних процедур, подальша деталізація яких не має сенсу. При цьому автоматизована система повинна зберігати цілісне уявлення, в якому всі складові її компоненти взаємопов'язані (принцип узгодженості).

Більшість методологій структурного аналізу і проектування ґрунтується на поданні моделей розроблюваних систем у вигляді діаграм. Найбільш популярні з них представлені в табл. 4.1.

Схематично застосування структурного підходу зображено на рис. 4.1.

Таблиця 4.1 – Методології структурного аналізу і проектування

Методологія	Тип моделі, яка розроблюється
SADT (Structured Analysis and Design Technique, методологія структурного аналізу і проектування)	Функціональна
DFD (Data Flow Diagrams,діаграми потоків даних)	Функціональна або компонентна
ERD (Entity-Relationship Diagrams,діаграми "сутність-зв'язок")	Інформаційна
Flowcharts (блок-схеми)	Поведінкова
EPC (Event-driven Process Chain, подієвий ланцюжок процесів)	Функціональна або поведінкова
BPMN (Business Process Model and Notation,модель і нотація бізнес-процесів)	Функціональна або поведінкова

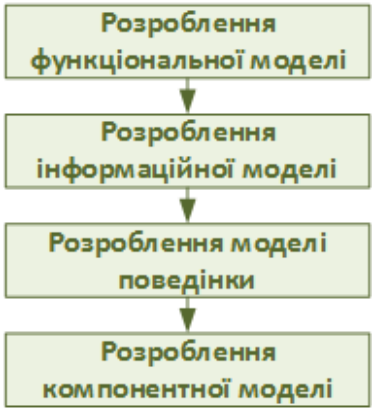


Рисунок 4.1 – Схема застосування структурного підходу

Спершу розробляється функціональна модель, за допомогою якої визначаються, аналізуються і фіксуються вимоги до складу та структури функцій системи, тобто визначається, для яких цілей розробляється система, які функції вона виконуватиме. На цій же моделі вказуються вихідна інформація, проміжні та підсумкові результати роботи системи. На основі інформаційних потоків визначається склад і структура необхідних даних, що зберігаються в системі (будується інформаційна модель). Далі, з урахуванням розроблених моделей, створюються процедури реалізації функції, тобто алгоритми обробки даних і поведінки елементів системи. На заключній стадії встановлюється розподіл функцій по підсистемах (компонентам), необхідне технічне забезпечення і будується модель їх розподілу по вузлах системи.

Приведена на рисунку схема не означає, що побудова моделей має строго відповідати зазначеному порядку – одна за одною. Як правило, розробка кожної наступної моделі починається ще до повного завершення розробки попередньої, а іноді – паралельно.

На стадії проектування моделі розширюються, уточнюються і доповнюються діаграмами, що відбивають технологію використання системи, її архітектуру, екранні форми і т. п.

Усі найбільш поширені методології структурного підходу базуються на ряді загальних принципів. В якості двох базових принципів використовуються:

- принцип "розділяй і володарюй" – принцип вирішення складних проблем шляхом їх розбиття на безліч менших незалежних завдань, легких для розуміння і вирішення;
- принцип ієрархічного впорядкування – принцип організації складових частин проблеми в ієрархічні деревоподібні структури з додаванням нових деталей на кожному рівні.

Виділення двох базових принципів не означає, що інші принципи є другорядними, оскільки ігнорування кожного з них може призвести до непередбачуваних наслідків (у тому числі і до провалу всього проекту). Основними з цих принципів є такі:

- принцип абстрагування – полягає у виділенні істотних аспектів системи і відволікання від несуттєвих;
- принцип формалізації – полягає в необхідності суворого методичного підходу до вирішення проблеми;
- принцип несуперечності – полягає в обґрунтованості та узгодженості елементів;
- принцип структурування даних – полягає в тому, що дані повинні бути структуровані і ієрархічно організовані.

4.3 СУТНІСТЬ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПІДХОДУ

Перша відмінність цих підходів один від одного полягає в принципах декомпозиції та структурної організації елементів (компонентів, модулів) системи. Згідно з цими принципами система являє собою структуру, що складається з чітко виражених модулів, пов'язаних між собою певними відносинами.

Другий вид декомпозиції – об'єктно-орієнтована. В рамках цього підходу система розбивається на набір об'єктів, відповідних об'єктам реального світу, взаємодіючих між собою шляхом посилки повідомлень.

Другою відмінністю є об'єднання в об'єкті як атрибутивних даних (характеристики, властивості), так і поведінки (функції, методи). У функціонально-орієнтованих системах функції і дані зберігаються (існують) окремо.

Третя відмінність двох підходів полягає в структурній організації всередині модулів системи. У структурному підході модуль складається з функцій, ієрархічно пов'язаних між собою відношенням композиції, тобто функція складається з підфункцій, підфункція з підпідфункцій і т.д. В об'єктно-орієнтованому підході ієрархія вибудовується з використанням двох типів відношень: композиції і спадкування (англ. IS A – це є). При цьому в об'єктно-орієнтованому підході «об'єкт-частина» може включатися відразу в кілька «об'єктів-ціле». Таким чином, модуль в структурному підході представляється у вигляді дерева, а в об'єктно-орієнтованому підході – у вигляді орієнтованого графа, тобто за допомогою більш загальної структури.

Найбільш популярними методологіями, що підтримують об'єктно-орієнтований підхід, зараз є:

- уніфікований процес (Unified Process, UP);
- екстремальне програмування (eXtreme Programming, XP);
- гнучке моделювання (Agile Modeling, AM).

Базовим засобом фіксації (документування) результатів проектування систем за допомогою цих методологій є уніфікована мова моделювання (Unified Modeling Language, UML).

4.3.1. Основні поняття, що використовуються в об'єктно-орієнтованому підході.

При розробці програмного забезпечення термін «об'єкт» вперше був введений Оле-Джоаном Далем, Бйорном Мюрхогом і Крістеном Нигардом з Норвезького обчислювального центру (м. Осло). Вони розробили мову Simula 67, створену на основі мови Algol-60 і призначену для

моделювання та опису складних систем. Однак по-справжньому широке впровадження цієї ідеї відбулося при розробці мови SmallTalk в 1990 р Аланом Кейем з Дослідницького центру фірми Xerox (м Пало-Альто). У SmallTalk використовувалися тільки об'єктно-орієнтовані конструкції.

Згідно них об'єкт – це абстракція реальної чи уявної сутності з чітко вираженими концептуальними межами, індивідуальністю (ідентичністю), станом і поведінкою.

Абстракція (лат. Abstractio – відволікання) – форма пізнання, заснована на уявному виділенні істотних властивостей і зв'язків предмета і відверненні від інших, приватних його властивостей і зв'язків. При цьому «істотне» і «приватне» повинні розглядатися з точки зору розв'язуваної задачі (предметної області). В об'єктно-орієнтованому підході абстракція – це модель суті, описує її властивості і поведінку.

Прикладами реальних (фізичних, відчутних) сутностей можуть служити поїзд, стрілочний перевід або інженер служби колії, і уявних – технологія проведення капітального ремонту колії або оптимальна траєкторія руху поїзда (режими і швидкість залежно від поточного положення поїзда на ділянці).

Індивідуальність – це властивість сутності, за допомогою якої її можна відрізнити від інших. Тобто, говорячи про об'єкт «поїзд», мається на увазі не узагальнене поняття поїзд, як те, що складається з локомотивів і вагонів, а конкретний вантажний потяг з номером 1025, вагою 4600 т, ведений електровозом змінного струму ВЛ80Т з серійним номером 027, що складається з чотиривісних піввагонів з конкретними номерами і т. д. У той же час ступінь абстракції з погляду розв'язуваної задачі може бути і вищою. Наприклад, при виконанні тягових розрахунків до графіку руху поїздів не потрібна інформація про серійні номери локомотивів і вагонів, тобто немає потреби у відмінності один від одного електровозів ВЛ80Т з серійними номерами 027 і 028.

Для концептуального угрупування однотипних об'єктів в об'єктно-орієнтованому підході використовується поняття «клас». Клас – це безліч об'єктів, що мають загальну структуру і поведінку. Таким чином, клас – це шаблон, на основі якого генеруються (створюються) однотипні об'єкти. Як синонім поняття «об'єкт» часто вживають поняття «екземпляр класу».

Кожен клас і відповідно об'єкт характеризуються строго певним набором атрибутів і методів. Поточні значення атрибутів чітко визначають поточний стан об'єкта. Набір методів і їх алгоритмічна реалізація визначають поведінку об'єкта (класу об'єктів).

Говорячи про об'єктно-орієнтований підхід, в першу чергу відзначають успадкування, інкапсуляцію і поліморфізм. Ці механізми і принципи проектування більш природно і повно реалізовані в об'єктно-орієнтованому підході в порівнянні зі структурним.

Спадкування – принцип, відповідно до якого знання про загальну категорію дозволяється застосовувати для більш вузької. Стосовно до класів це означає, що дочірній клас (вузька категорія) повністю включає в себе (успадковує) всі атрибути і методи, визначені в батьківському класі (загальній категорії). При цьому в дочірньому класі можуть бути визначені додаткові атрибути і методи. Наприклад, дочірній клас «коло» буде наслідувати від батьківського класу «геометрична фігура» всі атрибути (x, y – координати центру фігури, color – колір фону і т. д.) і всі методи (draw () – намалювати фігуру, move (dx, dy) – перемістити фігуру і т. д.), а також мати додатковий атрибут (r – радіус).

Інкапсуляція (інформаційна закритість) – принцип, відповідно до якого зміст внутрішньої структури елементів системи має бути прихованим один від одного. Цей принцип представляє обмін інформацією між об'єктами системи тільки в мінімально необхідному обсязі, обмеження доступу до атрибутів і методам об'єктів (класів) з боку інших об'єктів (класів) і повне приховування алгоритмічної реалізації методів від інших об'єктів (класів).

Поліморфізм – принцип побудови елементів моделі так, щоб вони могли приймати різні зовнішні форми або функціональність (поведінка) залежно від обставин. Наприклад, методи draw () (намалювати) або calculates () (розрахувати площе) для класів «коло» і «ромб», визначених шляхом успадкування атрибутів і методів батьківського класу «фігура», алгоритмічно повинні бути реалізовані по-різному.

З урахуванням наведених вище визначень сутність об'єктно-орієнтованого підходу до аналізу і проектування інформаційних систем полягає в декомпозиції системи на класи, які відповідають однотипним об'єктам предметної області, і побудові з них ієрархії у вигляді орієнтованого графа з використанням відношень композиції і успадкування.

4.3.2. Базові складові об'єктно-орієнтованого підходу.

Базовими складовими об'єктно-орієнтованого підходу є:

- уніфікований процес;
- уніфікована мова моделювання;
- шаблони проектування.

Уніфікований процес – це процес розробки програмного забезпечення (ПЗ), який забезпечує упорядкований підхід до розподілу завдань і обов'язків в організації-розробника. Уніфікований процес охоплює весь життєвий цикл ПЗ, починаючи з визначення вимог і закінчуючи супроводом, і являє собою узагальнений каркас (шаблон, скелет), який може бути застосований (спеціалізований) для розробки і супроводу широкого кола систем.

Невід'ємною частиною уніфікованого процесу є UML – мова (система позначень) для визначення, візуалізації і конструювання моделей системи у вигляді діаграм і документів на основі об'єктно-орієнтованого підходу. Слід зазначити, що уніфікований процес і UML розроблялися спільно.

На стадіях аналізу і проектування часто використовуються так звані шаблони (патерни) проектування. Шаблон – це іменована пара «проблема / рішення», що містить готове узагальнене рішення типової проблеми. Як правило, шаблон крім текстового опису містить також одну або декілька діаграм UML (наприклад, діаграми класів, послідовності і / або комунікації), графічно ілюструють склад і структуру класів, а також особливості їх взаємодії при вирішенні поставленої проблеми. Шаблони розробляються досвідченими професіоналами і є перевіреними, ефективними (часом оптимальними) рішеннями. Застосування шаблонів може різко скоротити витрати і підвищити якість розробки ПЗ.

.

ВИСНОВКИ

На відміну від структурного підходу об'єктно-орієнтований має ряд переваг.

Опис системи у вигляді об'єктів більше відповідає змістовному опису предметної області. Наприклад, при використанні структурного підходу БД повинна задовольняти вимогам нормалізації, відповідно до яких дані по одному і тому ж об'єкту (сутності з реального світу)

можуть зберігатися в кількох таблицях.

Сутності реального світу, як правило, володіють поведінкою, що в об'єктно-орієнтованому проектуванні відбивається за допомогою визначення методів класу. У структурному підході дані (атрибути) і алгоритми (методи) існують окремо один від одного.

Об'єднання атрибутів і методів в об'єкті (класі), а також інкапсуляція дозволяють домогтися більшої внутрішньої і меншою зовнішньої зв'язності між компонентами системи. Це полегшує вирішення проблем:

- адаптації системи до зміни існуючих або появі нових вимог;
- супроводу системи на різних стадіях життєвого циклу;
- повторного використання компонентів;

Об'єктно-орієнтований підхід дозволяє легше організувати паралельні обчислення, оскільки кожен об'єкт володіє власними значеннями характеристик (атрибутів) і поведінкою, за рахунок чого можна домогтися його автономної роботи.

CASE-засоби, що підтримують об'єктно-орієнтований підхід, на основі інформації про об'єкти дозволяють досягти більшого ступеня автоматизації кодогенерації. CASE-засоби, що підтримують структурний підхід, добре справляються з генерацією структур БД. Однак слід зазначити, що ця структура повинна задовольняти вимогам нормалізації. У зв'язку з тим автоматична кодогенерація (наприклад, екранів або функцій обробки даних) можлива лише в рідкісних випадках.

- 5.1 Основи уніфікованої мови моделювання UML
- 5.2 Проектування логічної моделі ІС і моделей баз даних
- 5.3 Проектування фізичної моделі ІС
- 5.4 Отримання схеми реляційної бази даних з діаграм класів
- Висновки

5.1 ОСНОВИ УНІФІКОВАНОЇ МОВИ МОДЕЛЮВАННЯ UML

Існує велика кількість інструментальних засобів, які використовуються для реалізації проекту ІС від етапу аналізу до створення програмного коду. Окремо виділяють так звані CASE-засоби верхнього рівня (upper CASE tools) і CASE-засоби нижнього рівня (lower CASE tools).

Серед основних проблем використання CASE-засобів верхнього рівня виділяють проблеми їх адаптації під конкретні проекти, оскільки вони жорстко регламентують процес розробки і не дають можливості організувати роботу на рівні окремих елементів проекту. Альтернативою їм може стати використання CASE-засоби нижнього рівня, але їх використання спричиняє інші проблеми – труднощі в організації взаємодії між командами, що працюють над різними елементами проекту.

Засобом, що дозволяє об'єднати ці підходи, є уніфікована мова об'єктно-орієнтованого моделювання (Unified Modeling Language – UML). До переваг мови UML можна віднести різноманітні інструментальні засоби, які як підтримують життєвий цикл ІС, так і дозволяють налаштувати і відобразити специфіку діяльності розробників різних елементів проекту.

Основними характеристиками об'єктно орієнтованої мови моделювання UML є:

- організація взаємодії замовника і розробника (груп розробників) ІС шляхом побудови репрезентативних візуальних моделей;
- спеціалізація базових позначень для конкретної предметної області.

Базовий набір діаграм UML міститься у великій кількості засобів моделювання. Однак у зв'язку з тим, що кожна прикладна задача має свої особливості і не вимагає всіх концепцій у кожному додатку, мова надає користувачам такі можливості, як:

- моделювання з використанням тільки засобів «ядра» для типових додатків;
- моделювання з використанням додаткових умовних позначень, якщо вони відсутні в «ядрі», або спеціалізація нотації і обмежень для даної предметної області.

Для підтримки моделювання різних етапів життєвого циклу ІС мова UML пропонує цілу сукупність діаграм.

У нотації мови UML визначені наступні види канонічних діаграм:

- варіантів використання (use case diagram);
- класів (class diagram);
- кооперації (collaboration diagram);
- послідовності (sequence diagram);
- станів (statechart diagram);
- діяльності (activity diagram);
- компонентів (component diagram);
- розгортки (deployment diagram);

Перелік цих діаграм і їх назв є канонічними в тому сенсі, що являють собою невід'ємну частину графічної нотації мови UML. Більше того, процес об'єктно-орієнтованого проектування нерозривно пов'язаний із процесом побудови цих діаграм. Сукупність побудованих у такий спосіб діаграм є самодостатньою в тому сенсі, що в них міститься вся інформація, яка необхідна для реалізації проекту складної системи (рис. 5.1).

Кожна з цих діаграм деталізує і конкретизує різні представлення про модель складної системи в термінах мови UML. При цьому діаграма варіантів використання являє собою найбільш загальну концептуальну модель складної системи, що є вихідною для побудови всіх інших діаграм. Діаграма класів, по своїй суті – логічна модель, що відбиває статичні аспекти структурної побудови складної системи.

Рисунок 5.1 - Класифікація груп діаграм UML 2.0

Діаграми кооперації і послідовностей являють собою різновид логічної моделі, що відображають динамічні аспекти функціонування складної системи. Діаграми станів і діяльності призначені для моделювання поведінки системи. Діаграми компонентів і розгортання служать для представлення фізичних компонентів складної системи і тому відносяться до її фізичної моделі. Крім графічних елементів, що визначені для кожної канонічної діаграми, на них може бути зображена текстова інформація, що розширює семантику базових елементів.

Далі будуть детальніше розглянуті перераховані діаграми із зазначенням їх призначення в процесі проектування ІС.

5.2 ПРОЕКТУВАННЯ ЛОГІЧНОЇ МОДЕЛІ ІС І МОДЕЛЕЙ БАЗ ДАНИХ

Однією з діаграм, що застосовуються на етапі проектування логічної моделі ІС, є діаграма варіантів використання (діаграма прецедентів, use case diagram), призначена для побудови на концептуальному рівні моделі того, як функціонує система в навколишньому середовищі.

Основними елементами для побудови моделі прецедентів на діаграмі є:

- Актор (actor) – елемент, що позначає ролі користувача, який взаємодіє з певною сутністю;
- Прецедент – елемент, що відображає дії, що виконуються системою (в т.ч. із зазначенням можливих варіантів), які призводять до результатів, спостережуваним акторами.

Між прецедентами в моделі можуть бути встановлені зв'язки, такі, як:

- Узагальнення (Generalization) – вказує спільність ролей;
- Включення (include) – вказує взаємозв'язок декількох варіантів використання, базовий з яких завжди використовує функціональне поведінка пов'язаних з ним прецедентів;
- Розширення (extend) – вказує взаємозв'язок базового варіанту використання і варіантів використання, які є його спеціальними випадками.

Приклад діаграми варіантів використання представлений на рис. 5.2.

Більш детально описати бізнес-процеси дозволяють діаграми діяльності та діаграми послідовностей.

Діаграма діяльності (activity diagram) – діаграма, що використовується при моделюванні бізнес-процесів, на якій представлено розкладання на складові частини деякої діяльності, а саме: скоординованого виконання окремих дій і вкладених видів діяльності, які з'єднуються між собою потоками від виходів одного вузла до входів іншого, із зазначенням їх виконавців.

Приклад діаграми діяльності представлений на рис. 5.3.

Розроблені на етапі побудови моделей бізнес-прецедентів діаграми видів діяльності можуть коригуватися внаслідок виявлення нових подробиць в описі бізнес-процесів об'єкта автоматизації на етапах аналізу і проектування.

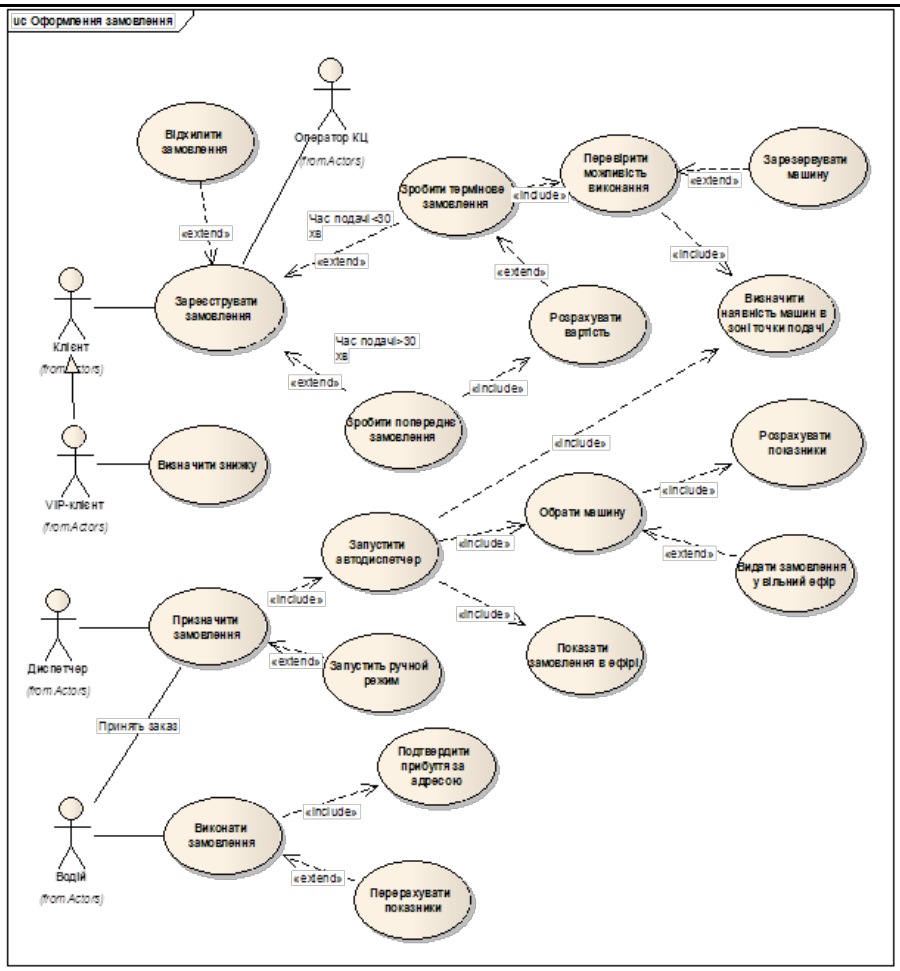


Рисунок 5.2 – Приклад діаграми прецендентів для інформаційної системи Оформлення системи

Діаграма послідовності (sequence diagram) – діаграма, що відображає впорядковані за часом прояви взаємодії об'єктів.

На діаграмі даного типу зліва направо поміщаються основні елементи: об'єкти; вертикальні лінії (lifeline), що моделюють протягом часу при виконанні дій об'єктом; стрілки, що визначають дії, що виконуються об'єктом.

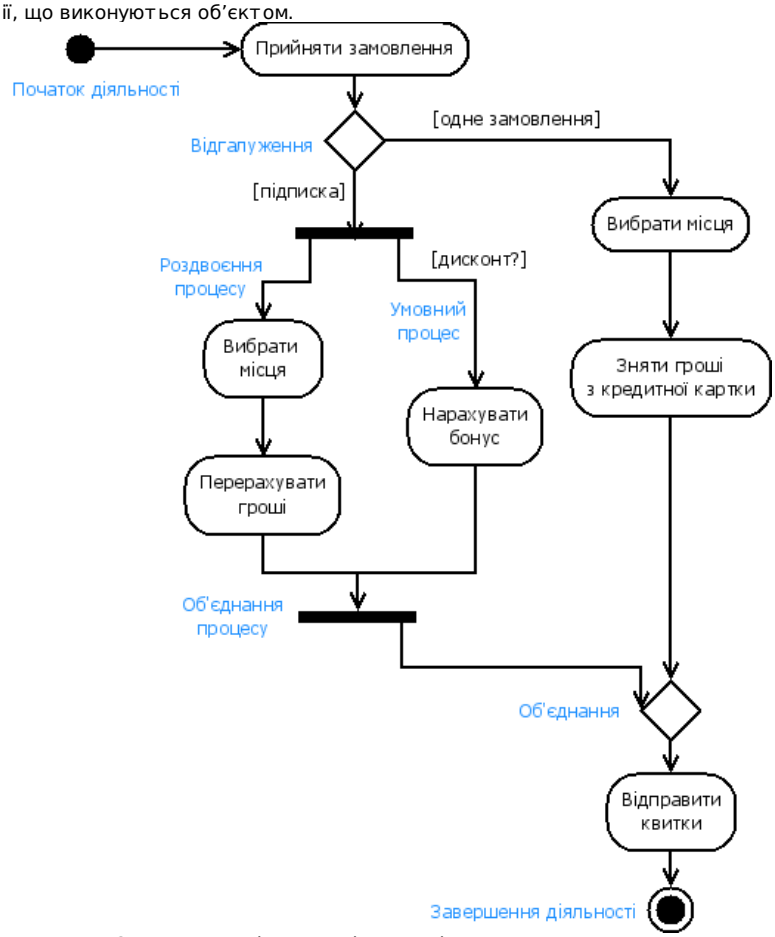


Рисунок 5.3 – Приклад діаграми діяльності

Приклад діаграми послідовності представлений на рис. 5.4.

У результаті побудови діаграм на цьому етапі розроблені докладні описи дій фахівців по

впровадженню ІС, які необхідні для забезпечення її функціональності.

На етапі формування вимог розробляється модель системних прецедентів, на якій для внутрішніх і зовнішніх виконавців вказуються їх конкретні обов'язки з використанням ІС. Моделі системних прецедентів розробляються на основі бізнес-моделей, побудованих на попередньому етапі, однак при цьому необхідно детально описати прецеденти з визначеннями використовуваних даних і зазначенням послідовності їх виконання, тобто докладно описати реалізацію функцій проектованої системи

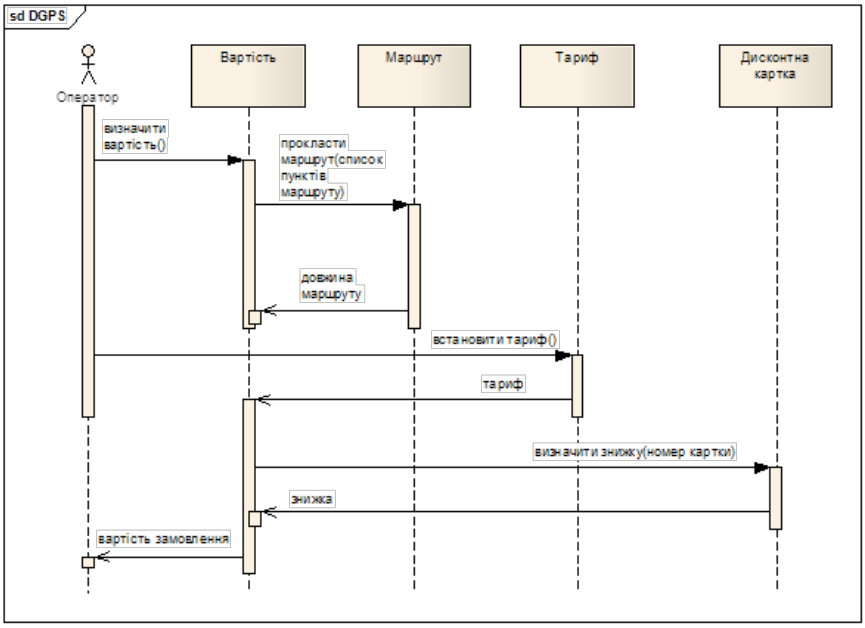


Рисунок 5.4 – Приклад діаграми послідовності

На етапі аналізу вимог і попереднього проектування системи на основі побудованих моделей системних прецедентів будуються діаграми класів системи.

Діаграма класів, будучи логічним поданням моделі, представляє детальну інформацію про структуру моделі системи з використанням термінології класів об'єктно-орієнтованого програмування, а саме: про внутрішній устрій системи (про архітектуру системи). На діаграмі класів можуть бути вказані внутрішня структура і типи відносин між окремими об'єктами і підсистемами, що призводить до розвитку концептуальної моделі системи.

Клас в мові UML позначає деяку множину об'єктів, що володіють однаковою структурою і взаємозв'язками з об'єктами інших класів. У діаграмах класів системи вказуються об'єкти з моделі системних прецедентів з їх описом і вказівкою взаємозв'язків між класами.

Синтаксис діаграм класів є ефективним засобом структурування вимог до елементів проектованої системи, до їх даних, інтерфейсів, функціональності.

Приклад діаграми класів представлений на рис. 5.5.

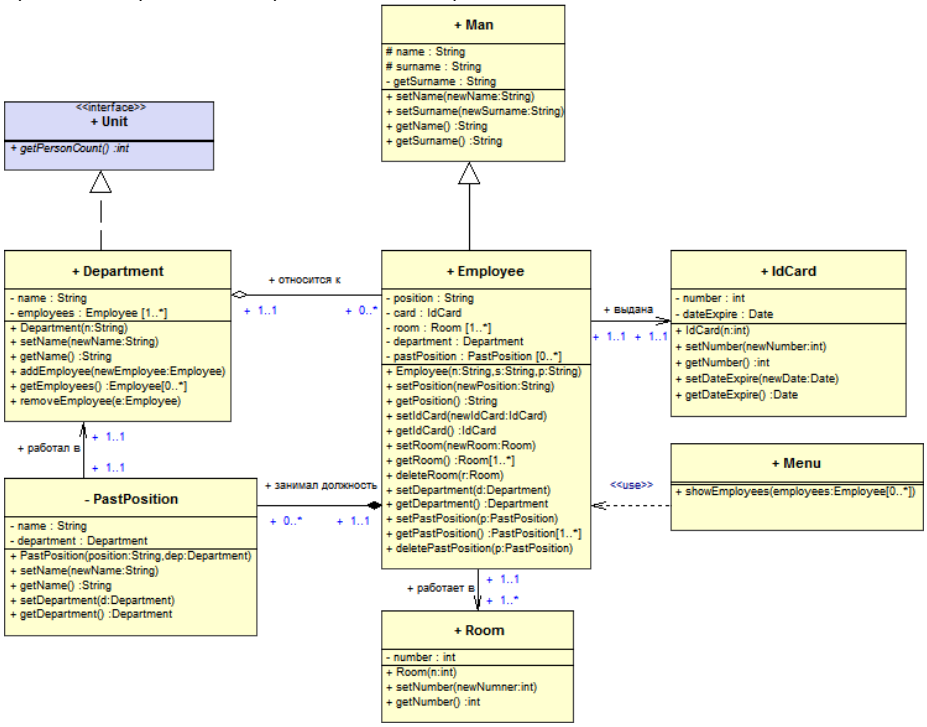


Рисунок 5.5 – Приклад діаграми класів

На цьому етапі проектування докладно описані склад і функції системи відповідно розробленим бізнес-моделям, що дає впевненість у відповідності проектованої системи вимогам замовника.

На наступному етапі елементи розроблених моделей класів відображаються в елементи моделей бази даних і додатків, а саме:

- Класи – в таблиці;
- Атрибути – в стовпці;
- Типи – в типи даних СУБД;

- Асоціації – у зв’язки між таблицями (у тому числі, створюючи додаткові таблиці зв’язків);
- додатки – в класи з визначеними методами і атрибутами, звя’заними з даними в базі.

У моделі бази даних для кожного простого класу формується таблиця, яка включає стовпці, поставлені у відповідність атрибутам класу.

Класи підтипів можуть відображатися в таблиці декількома способами:

- у разі відображення однієї таблиці на клас окрема таблиця формується для кожного класу з встановленням наступних зв’язків;
- у разі відображення однієї таблиці на суперклас для суперкласу створюється таблиця, а потім у таблиці підкласів розміщуються стовпці для атрибутів суперкласу;
- у разі відображення однієї таблиці на ієрархію формується єдина таблиця, в якій розташовані атрибути суперкласу і всіх підкласів з додаванням додаткових стовпців для визначення вихідних таблиць підкласів.

Мова UML для розробки проекту БД пропонує спеціальний профіль Profile for Database Design, в якому використовуються основні компоненти діаграм, такі, як: таблиця, стовпець, ключі, зв’язки, домен, процедура і т. д. На діаграмах також можна вказувати додаткові характеристики стовпців і таблиць: обмеження, тригери, типи даних і т. д. У результаті цього етапу проект бази даних і додатків системи стає детально описаним.

5.3 ПРОЕКТУВАННЯ ФІЗИЧНОЇ МОДЕЛІ ІС

Наступний етап проектування системи включає в себе доповнення моделі баз даних і додатків діаграмами їх розміщення на технічних засобах.

На даному етапі розглядаються такі поняття UML, як:

- компонент – елемент фізичного представлення системи, який реалізує певний набір інтерфейсів;
- залежність – зв’язок між двома елементами, що позначає ситуацію, при якій зміна одного елемента моделі тягне за собою зміну її іншого елемента;
- процесор і пристрій – вузол, що виконує і не виконує обробку даних відповідно;
- з’єднання – зв’язок між процесорами і пристроями.

Найбільш повно особливості фізичного представлення системи на мові UML дозволяють реалізувати діаграми компонентів і діаграми розгортання.

Діаграма компонентів (component diagram) відображає ієрархію підсистем, структурних компонентів і залежностей між ними. Фізичними компонентами виступають бази даних, виконувані файли, додатки, бібліотеки, інтерфейси ІС і т. д. У разі використання діаграми компонентів для відображення внутрішньої структури компонентів, інтерфейси складеного компонента делегуються в певні інтерфейси внутрішніх компонентів.

Основними цілями побудови діаграм компонентів є:

- визначення архітектури проектованої системи;
- побудова концептуальної і фізичної моделей баз даних;
- представлення структури вихідного і специфіки виконуваного коду системи;
- багаторазове використання певних фрагментів програмного коду.

Приклад діаграми компонентів наведено на рис. 5.6.

Рисунок 5.6 – Приклад діаграми компонентів

Для опису апаратної конфігурації ІС застосовують діаграму розгортання (deployment diagram) (рис. 5.7).

За допомогою діаграми розгортання моделюють фізичний розподіл різних програмних, інформаційних, апаратних компонентів системи по комплексу технічних засобів. Особливу увагу на діаграмі розгортання приділяється відображенню того, які використовуються апаратні компоненти («вузли» – сервери баз даних і додатків, web-сервер), які програмні компоненти («артефакти» – бази даних, web-додатки) працюють на кожному з них і як частини комплексу з’єднані один з одним. Таким чином, у разі проектування складної ІС її необхідно розділити на частини і досліджувати кожну частину окремо. Існують два основних способи розбиття ІС на такі підсистеми:

- Структурна (функціональна) декомпозиція;
- Об’єктна (компонентна) декомпозиція.

Характерною особливістю об’єктної декомпозиції є виділення об’єктів (компонентів), взаємодіючих між собою, виконуючих певні функції (методи) об’єкта.

При проектуванні ІС мову UML використовують для візуального моделювання системи при її розбитті на об’єкти. У разі функціональної декомпозиції ІС при проектуванні використання UML недоцільно.

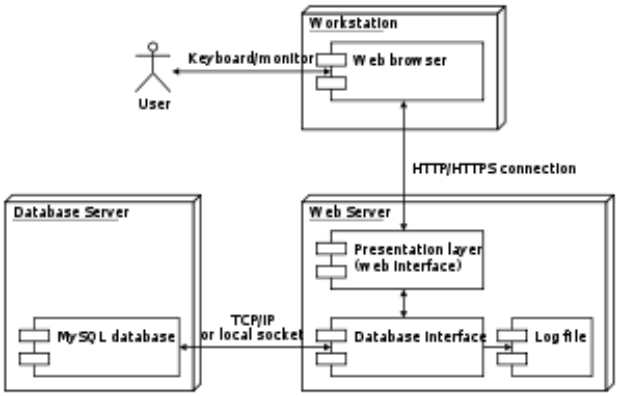


Рисунок 5.7 – Приклад діаграми розгортання

5.4 ОТРИМАННЯ СХЕМИ РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ З ДІАГРАМ КЛАСІВ

Взагалі кажучи, перехід від діаграмного подання концептуальної схеми бази даних до її реляційної схеми не залежить від різновиду використовуваних діаграм. Зокрема, методика, розроблена для класичних діаграм «Сутність-Зв'язок» (Entity-Relationship), практично завжди придатна для діаграм класів UML.

Рекомендація 1. Перш, ніж визначати в класах операції, подумайте, що можна зробити з цими визначеннями в середовищі цільової РСУБД. Якщо в цьому середовищі підтримуються збережені процедури, то, можливо, деякі операції можуть бути реалізовані саме за допомогою такого механізму. Але якщо в середовищі РСУБД підтримується механізм визначених користувачами функцій, він може виявитися більш відповідним.

Рекомендація 2. Пам'ятайте, що порівняно ефективно в РСУБД реалізуються тільки асоціації видів "один-до-багатьох" і "багато-до-багатьох". Якщо у створеній діаграмі класів є асоціації "один-до-одного", то слід задуматися над доцільністю такого проектного рішення. Реалізація в середовищі РСУБД асоціацій з точно заданими кратностями ролей можлива, але вимагає визначення додаткових тригерів, виконання яких знизить ефективність.

Рекомендація 3. Для технології реляційних БД агрегатні і особливо композитні асоціації є неприродними. Подумайте про те, що ви хочете отримати в реляційної БД, оголосивши деяку асоціацію агрегатною.

Рекомендація 4. У специфікації UML говориться, що, визначаючи односпрямовані зв'язки, ви можете сприяти ефективності доступу до деяких об'єктів. Для технології реляційних баз даних підтримка такого оголошення викличе додаткові накладні витрати і тим самим знизить ефективність.

ВИСНОВКИ

Метод об'єктно-орієнтованого моделювання передбачає послідовне виконання двох етапів: об'єктно-орієнтованого аналізу та об'єктно-орієнтованого проектування. Тому термін "об'єктно-орієнтоване моделювання" еквівалентний терміну "об'єктно-орієнтований аналіз і проектування"

Аналіз – широке поняття. Його зміст детальніше відображають терміни аналіз системних вимог (тобто дослідження вимог до майбутньої програмної системи) та об'єктний аналіз (тобто дослідження об'єктів предметної області).

При цьому під предметною областю розуміють ту частину реального світу, що має істотне значення чи безпосереднє відношення до процесу функціонування програми. Іншими словами: предметна область містить у собі тільки ті об'єкти і взаємозв'язки між ними, які необхідні для опису вимог і умов розв'язання деякої задачі.

У загальному випадку виділення базових об'єктів (чи компонентів) предметної області є нетривіальною задачею. Складність виявляється у неформальному характері процедур чи правил, які можна застосовувати з цією метою. Окрім того, таку роботу необхідно виконувати спільно з фахівцями чи експертами, що добре знають предметну область.

У процесі проектування головну увагу звертають на концептуальні рішення, які забезпечують виконання системних вимог, а не на питання реалізації. У процесі об'єктно-орієнтованого проектування визначають програмні об'єкти та способи їхньої взаємодії і/або схеми баз даних.

Під час аналізу системних вимог необхідно з'ясувати потреби замовника, аналізуючи отриману інформацію від керівництва компанії та майбутніх користувачів системи. Під час аналізу необхідно визначити:

- функціональні вимоги до системи (або бізнес-процеси), тобто встановити варіанти використання програмної системи для реалізацій конкретних функцій чи дій у даній предметній області ("визначити те, що система має робити");
- потоки даних для кожного бізнес-процесу;
- границі системи;
- користувачів системи та процеси їхньої взаємодії з системою.

У результаті аналізу, зазвичай, оформляють словник (або глосарій) предметної області (містить текстовий опис термінів, сутностей, користувачів тощо) і технічне завдання, у якому сформульовано функціональні та нефункціональні вимоги до системи. До нефункціональних вимог відносяться питання надійності, зручності використання, продуктивності, можливості супроводу програм, питання безпеки, проектні та апаратні обмеження тощо.

Технічне завдання є гарантією єдиного трактування вимог замовниками і проектувальниками. Воно дає змогу також вирішувати спірні питання з приводу функцій системи, що виникають у процесі її створення.

Не можна сказати, що проектування баз даних на основі семантичних моделей у кожному разі прискорює і / або спрощує процес проектування. Все залежить від складності предметної

області, кваліфікації проектувальника і якості допоміжних програмних засобів. Але в кожному разі етап діаграмного моделювання забезпечує наступні переваги:

- на ранньому етапі проектування до прив'язки до конкретної РСУБД проектувальник може виявити і виправити логічні огріхи проекту, керуючись наочним графічним представленням концептуальної схеми;
- остаточний вигляд концептуальної схеми, отриманої безпосередньо перед переходом до формування реляційної схеми, а може бути, і проміжні версії концептуальної схеми повинні стати частина документації цільової реляційної БД; наявність цієї документації дуже корисно для супроводу і особливо для зміни схеми БД у зв'язку зі зміненими вимогами;
- при використанні CASE-засобів концептуальне моделювання БД може стати частиною всього процесу проектування цільової інформаційної системи, що може сприяти правильної структуризації процесу, ефективності та підвищенню якості проекту в цілому.

Хочу ще раз підкреслити, що в контексті проектування реляційних БД структурні методи проектування, засновані на використанні ER-діаграм, і об'єктно-орієнтовані методи, засновані на використанні мови UML, розрізняються, головним чином, лише термінологією. ER-модель концептуально простіше UML, в ній менше понять, термінів, варіантів використання.

Мова UML належить об'єктному світу. Цей світ набагато складніше реляційного світу. Оскільки UML може використовуватися для уніфікованого об'єктно-орієнтованого моделювання всього, що завгодно, в ньому міститься маса різних понять, термінів і варіантів використання, абсолютно надлишкових з точки зору проектування реляційних БД. Якщо виокремити з загального механізму діаграм класів те, що дійсно потрібно для проектування реляційних БД, то ми отримаємо в точності ER-діаграми з іншого нотацією і термінологією.

6.1 ОСНОВИ ГНУЧКОГО МОДЕЛЮВАННЯ

Відповідно водоспадній (каскадній) моделі життєвого циклу розробка програмного продукту проходить через ряд етапів:

- збір вимог;
- їх аналіз;
- створення архітектури;
- створення дизайну системи;
- кодування;
- тестування;
- викладка;
- експлуатація.

Послідовне виконання зазначених етапів гарантує отримання добротної інформаційної системи при стабільності вхідних умов та умов експлуатації.

Для чого потрібна методологія гнучкої розробки?

По-перше, це прискорення виведення продукту на ринок. Якщо необхідно розробити систему швидше, потрібно робити це відповідно до Agile. Дуже простий приклад. Є дві компанії, у них приблизно однаковий бізнес. Одна пише ТЗ, потім проектує систему і малює дизайн – це водоспадна модель, на розробку якої може піти кілька місяців. У другій компанії, що працює по Agile, до цього часу може бути вже запущений сайт, випущено ПЗ, вона почне заробляти гроші і захоплювати ринок, що найголовніше.

По-друге, управління змінами в пріоритетах. Якщо розроблюється проект тривалістю кілька місяців, то у вас обов'язково зміняться вимоги. Якщо говорити про комерційну розробку, то проблема в тому, що ми, програмісти, аналітики та дизайнери, ніколи не знаємо, що потрібно не тільки замовнику, який нам сплачує, але і користувачам. Зазвичай всі підходять до питання так: поки користувач не спробує функціонал сайту або програми, ви не знаєте, чи потрібен він чи ні.

По-третє, поліпшення взаємодії між IT і бізнесом. Це головний біль, особливо для великих компаній, адже у бізнесу періодично змінюються вимоги, кожен говорить своєю мовою. В результаті сторони один одного не розуміють.

Відповіддю на всі ці виклики з'явився Маніфест гнучкої розробки ПЗ.

6.2 МАНІФЕСТ ГНУЧКОЇ РОЗРОБКИ

6.2.1. Цінності.

Маніфест складається з декількох частин. Перша частина називається «Цінності» (Values). Це чотири «зважування»:

- Якщо ви хочете побудувати гнучкий процес, вам потрібно взаємодіяти і спілкуватися між собою. У чому це виражається розглянемо нижче на прикладі Scrum. При цьому ви можете

(і обов'язково будете) використовувати якісь інструменти і процеси, наприклад, трекери – JIRA, Redmine і т.д. Але ваша робота повинна спиратися на різні мітинги, зустрічі і взаємодію, а не на налаштування трекерів.
- Працюючий продукт, який ми робимо, набагато важливіше, ніж документація по ньому. Вище було наведено приклад з двома компаніями: у однієї є готовий продукт, який можна дати користувачам, замовнику, захопивши ринок; а друга пише ТЗ, малює макети і т. д. Вся ця документація, яку користувач не може застосувати по причини не готовності продукту, не приносить цінності цьому користувачеві. Якщо ми навчимося працювати, мінімізуючи ці кроки, або роблячи їх невеликими шматочками, то у нас вийде більш гнучкий процес.
- Співпраця та взаємодія з замовником важливіше жорстких контрактних обмежень. Зазвичай підписується договір, в якому зазначено, що до конкретної дати за певну суму розробник зобов'язується виконати обумовлений обсяг робіт. Природно, до договору прикладається ТЗ. Тобто фіксується час, обсяг робіт і терміни. Це називається Fixed Price. Такий підхід не дуже хороший, якщо ви хочете працювати на довгострокову перспективу і бути гнучкими. У цьому випадку правильніше вибудовувати партнерські відносини з замовником. Якщо говорити про контрактні оформлення, то зазвичай це виливається в контракти за схемою «час – матеріали», коли розробнику просто оплачується витрачений час. Найголовніше, що тут починається пошук партнерства і ситуації Win-Win, коли перемагає і замовник, і його підрядник.
- Готовність до змін в зважуванні зі проходженням первинним планом.

У Agile є план, оцінки і прогнози. Але якщо у вас є якийсь початковий план для річного проекту, а ви через три місяці вже надали якусь версію продукту, користувачі його пощупали, ви зняли метрики, подивилися, що і як вони використовують, дізналися щось нове, то після цього початковий план можна майже повністю поміняти.

Таким чином, в Agile існує чотири цінності:

- люди і взаємодія між ними;
- робочий продукт;
- співпраця і вибудовування партнерських відносин із замовником;
- готовність до змін.

6.2.2. Принципи Agile.

Цінності зумовлюють 12 принципів Agile:

1. *Найвища цінність* – це задоволення потреб замовника завдяки регулярній і максимально ранній поставці цінного для нього ПЗ. Якщо замовник хоче отримати від нас великого слона, але ми можемо дати йому частину цього слона не через рік, а через три місяці, потім ще через три місяці ще одну частину, а після щоміся ця видавати шматочки, то чим частіше ми це будемо робити і чим раніше, тим краще.
2. Ми завжди *готові змінювати вимоги*, навіть на пізніх стадіях проекту, якщо дізнаємося щось нове. Таким чином, ми створюємо бізнесу або зовнішньому замовнику конкурентну перевагу. Припустимо, працюють дві компанії: один написала ТЗ і за рік зробила продукт, а ми зробили концепцію продукту (неважливо, в якому вигляді) і поступово його викочуємо і розгортаємо. Тоді наш продукт буде більше відповідати вимогам замовників, користувачів і ринку в цілому.
3. При використанні Agile *працюючий продукт випускають максимально часто*. У маніфесті прописані терміни – від кількох тижнів до кількох місяців. Насправді це тиждень / місяць, якщо ви використовуєте Scrum. А якщо роблять якийсь веб-проект, то зазвичай використовують одну з варіацій Kanban, значить, релізи можна робити щодня. У HeadHunter зазвичай щодня виходить кілька релізів, що створює великі проблеми для наших конкурентів. Це можуть бути правки багів, але ми вміємо додавати щось цінне для наших користувачів.
4. *Бізнес обов'язково повинен працювати разом з програмістами*, допомагати їм зрозуміти специфіку даного ринку. Найбільш частою проблемою, з мого досвіду, є недоступність бізнесу – коли розробник не може отримати у співробітника потрібну інформацію. При використанні Agile важливо уникати виникнення подібних ситуацій.
5. *Команда – один з наріжних каменів* Agile. Найкращих результатів досягає команда замотивованих професіоналів. У Agile керівник передусім має створювати умови для команди і забезпечувати всебічну підтримку, проводити коучинг, стежити за атмосферою в колективі.
6. Є багато досліджень, які показують, що *краще спілкування – лицем до лиця*. Причому бажано, щоб було якесь засіб візуалізації, на якому можна писати: аркуш паперу, дошка зі стікерами і т. д. Найпростіший і ефективний спосіб дізнатися вимоги клієнта, замовника або користувача – поговорити з ними.
7. *Працюючий продукт*. Ступінь готовності проекту повинна вимірюватися не словами, про те, що ТЗ вже написано і 50% макетів намальовано, а кількістю функціоналу, випущеного в production.
8. У Agile важливий *ритм, постійні поліпшення*. Бізнес і програмісти завжди повинні мати можливість робити процес стійким, постійно його покращувати.
9. У вас повинна бути хороша *гнучка архітектура*, в яку можна додавати різні елементи і при необхідності легко їх змінювати. І якщо команда не приділятиме максимум уваги технічною якістю (писати хороший код, використовувати інженерні практики, автоматизувати процеси), то ніякого Agile у вас не буде.
10. *Простота*. Вона проявляється в технічній складовій, в дизайні. Це один з принципів екстремального програмування. Простота дуже важлива також з точки зору випуску продукту: коли ви хочете «нарізати» того «слона», краще почати з простої частини.
11. *Менеджер* (керівник, Scrum-коуч, Agile-коуч) в команді змінює свою роль: він не стільки займається організацією процесу, скільки *вчить команду*, тому *команда повинна бути самоорганізованою*. Є спеціальні стратегії, як з групи людей зробити самоорганізовану команду.
12. Команда повинна *постійно аналізувати свою роботу*, процеси: що вийшло, як вони цього добилися, і постійно покращувати організацію робіт.

Отже, у нас виходить піраміда, що складається з чотирьох цінностей, на яких збудовано 12 принципів. Тепер з'являються конкретні практики.

6.2.3. Практики.

Одна із цінностей Agile говорить: ми повинні вибудовувати робочий процес, налагоджуючи взаємодію та комунікації між людьми. Це виливається в практичні кроки. Наприклад, у ранковий стендап, коли команда усно синхронізує свою діяльність на майбутній день. Можна замість стендапів кожному писати звіт про зроблене вчора, але це вже буде не Agile, бо виникає потік малозначимої документації. Які ж практики найчастіше використовуються в компаніях, практикуючих гнучку розробку?

1. На першому місці стендап. Практика проста: кожен день в певний час команда збирається і синхронізує свою діяльність.

13. На другому місці планування ітерацій, коли команда планує обсяг робіт на найближчу ітерацію. Це до питання про те, що в Agile немає планування. Якщо ітерація йде два тижні, то слід намагатися зробити план, декомпозицію, розбити бізнес-завдання на завдання технічні.
14. Unit-тестування – одна з найпростіших інженерних практик, яку можна досить швидко почати використовувати. При наявності проблем з якістю близько 60-70% проблем можна відловити unit-тестуванням.
15. Планування релізів. Тут вже планується великими шматками – що і коли випустимо. Якщо мова йде про Scrum, то вимірюємо великими мазками, в спрінтах.

На рис. 6.1 графічно представлено ітеративне виконання робіт.

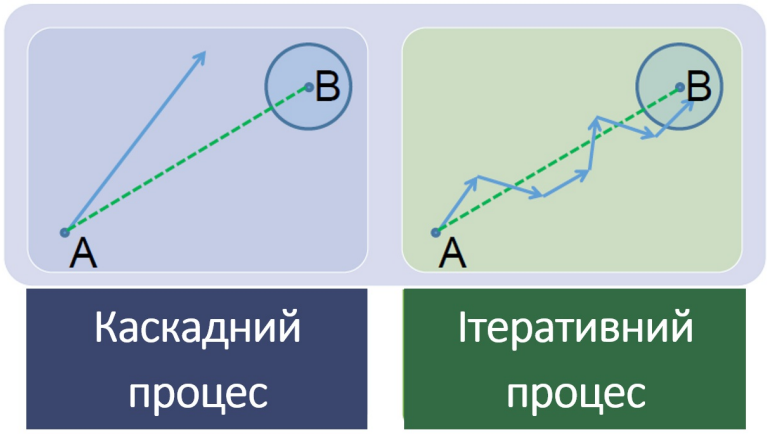


Рисунок 6.1 – Ітеративність виконання робіт з проектуванню ІС

Нам треба дійти з точки А в точку Б. «Дійти» – означає «отримати зворотній зв'язок». Водоспадна модель – фактично, один великий крок. Тобто я кудись прийшов, випустив на ринок продукт, і тільки після цього можу зрозуміти, чи то я зробив. Ітеративна модель – це серія кроків. Я роблю перший крок, знімаю метрики, що у мене використовується в продукті, потім коригую подальші кроки. Нехай я прийду не в точку Б, але опинюся в її околицях.

При комерційній розробці у нас постійно змінюються умови: виходять нові закони, змінюються бізнес-вимоги, конкуренти раптом випускають щось, що ми повинні зробити краще, ніж вони. У підсумку виходить, що ми з точки А повинні дійти не в точку Б, а в точку В. Ітеративна модель передбачає, що після випуску першого релізу ми знімаємо метрики, щось переробляємо, робимо наступний реліз, і т. д. І на якомусь етапі розуміємо, що конкурент випустив якусь фічу, і нам потрібно йти не туди. У цей момент ми можемо зупинитися, прийняти інші вимоги і почати рухатися в точку В. При високої мінливості умов у процесі створення продукту ви весь час будете дізнаватися щось нове. І в цьому одна з переваг ітеративної моделі. Щоб працювати итеративно і інкрементально, потрібно діяти трохи інакше. У розробника в голові повинна бути якась концепція, яку він поступово опрацьовує.

На відміну від водоспадного підходу в Agile слід діяти інакше: є команда і фіксовані відрізки часу (на прикладі Scrum), наприклад, двотижневі ітерації. Виходячи з цього, ми плануємо обсяг робіт, який ми можемо виконати за цей час. З позиції Agile наш проект тим успішніше, чим більше ми поставили цінностей замовнику.

6.3 ОСНОВИ SCRUM

6.3.1. Підхід.

Найбільш розвинені представники гнучкої методології – Scrum, екстремальне програмування (XP), DSDM, Crystal, FDD, Kanban. Більше половини компаній, що застосовують Agile, використовують Scrum. На другому місці комбінація Scrum і XP, коли беруться управлінські практики з Scrum і додаються інженерні. Комбінація Scrum і Kanban використовується приблизно в 8% компаній. Зараз це один із трендів, мода. Назва Scrum прийшла з регбі і перекладається як «сутичка». Простіше кажучи, при виникненні спірної ситуації команди шикуються, вкидається м'ячик, і потрібно один одного перештовхати. До розробки продукції цей термін вперше застосували в 1980-х роках два японці – Хіротака Такеуті і Ікудзіро Нонака. Вони проаналізували, як різні компанії створюють свої продукти. Причому навіть не ПЗ, а всіляку техніку та електроніку. Автори розділили виявлене підходи на три типи (рис. 6.2).

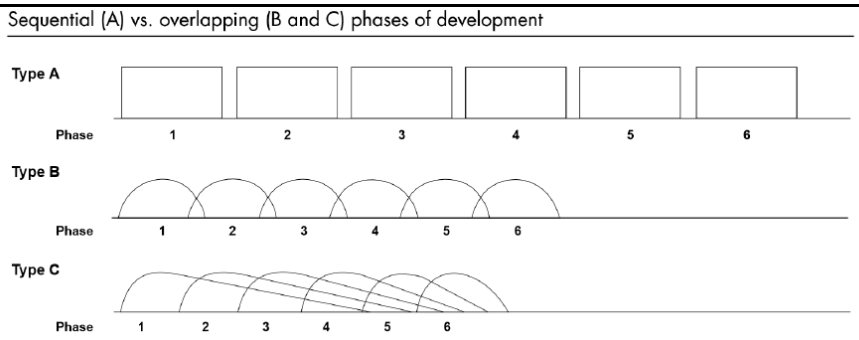


Рисунок 6.2 – Підходи до проектування об'єктів

Перший тип (A): у нас є фаза розробки, все жорстко, послідовно і добре.

Другий тип (B): зв'язки перетинаються, є можливість отримати зворотній зв'язок. Програміст запрограмував щось, програмує ще, віддає тестувальнику, тестувальник дає свої коментарі, програміст встигає їх обробити до завершення своєї роботи.

Третій тип: кожна фаза перетинається більш ніж з однією іншою фазою. Програміст програмує ще щось, а його перші завдання тестуються, викладаються в production, з них знімаються метрики, і програміст можу внести зміни.

Тип C двоє японців порівняли з ситуацією в регбі. Чому? Сенс гри в тому, щоб взяти м'яч і донести його до лінії поля противника, долаючи опір. Але в регбі можна кидати м'яч вперед. Коли ти біжиш і розумієш, що зараз тебе покладуть додолу, то м'яч можна віддати назад члену своєї команди. Він його ловить і біжить далі. Якщо не може подолати опір, то біжить назад.

Сучасний Scrum створений двома айтішниками – Кеном Швабером і Джефом Сазерлендом. Вигляд загальної схеми Scrum приведений на рис. 6.3.

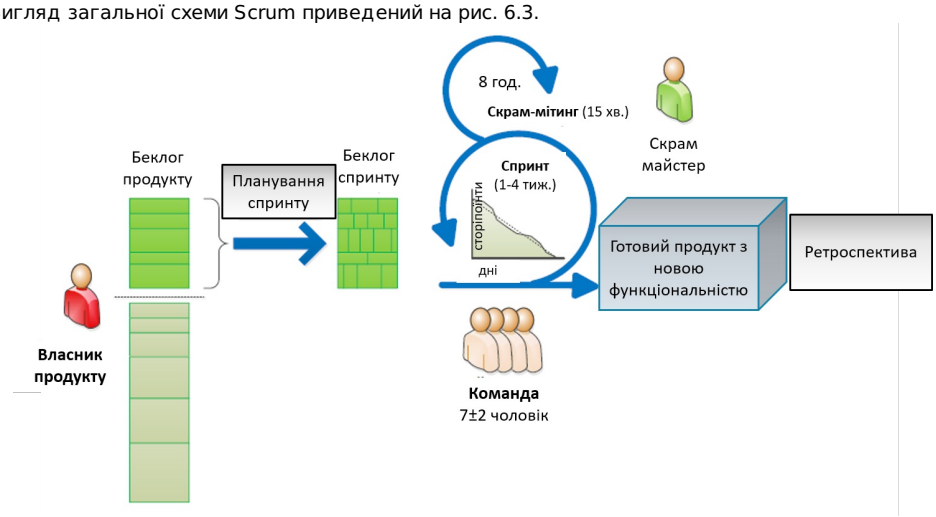


Рисунок 6.3 – Загальна схема Scrum

Отже, ми хочемо робити якийсь продукт. Він розрізаний на окремі шматочки, які називаються беклогом (backlog) продукту. Це вимога. Тобто у нас немає технічного завдання або якогось обширного документа, який все заздалегідь описує. Звичайно чим важливіше якісь вимоги, тим якісніше вони розбиті й описані. Цим займається власник продукту (product owner).

Середньостатистична команда складається з семи осіб (плюс-мінус дві особи). Якщо набагато менше, то, швидше за все, в команді не вистачає якихось фахівців, тому що мається на увазі, що Scrum-команда може самостійно зробити з беклога готовий продукт з новою функціональністю.

6.3.2. Компоненти Scrum.

Ролі. Scrum-команда складається з команди розробки, власника продукту і Scrum-майстра (рис. 6.4).

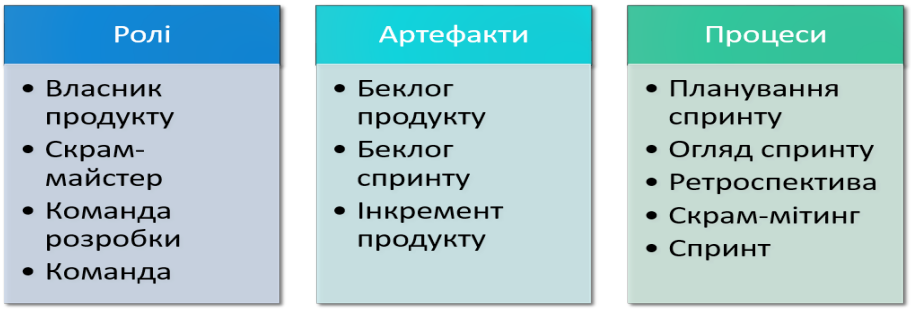


Рисунок 6.4 – Компоненти Scrum

Чому ми говоримо, що Scrum - це гнучкий Agile-фреймворк? Тому що він безпосередньо реалізує цінності та принципи, описані на початку публікації. Команда в Scrum повинна бути самоорганізована, а Scrum-майстер допомагає їй такою стати, вчить, як можна швидше і якісніше з елементів беклога створити інкремент продукту – нову версію софта. Scrum-майстер відповідає за те, щоб всі процеси працювали, щоб учасники розуміли, навіщо і як це робиться.

Власник продукту, product manager – людина, відповідальна за максимізацію цінності продукту, він відповідає за продукт.

Команда розробки повинна складатися з мотивованих професіоналів, які можуть протягом кожного спринту робити поставку, тобто на підставі вимог створювати готовий продукт.

Процеси. Щоденний скрам – це зустріч, на якій члени команди розповідають, що зроблено, з якими проблемами зіткнулися, що планується зробити найближчим часом, щоб кожен розумів, хто і що робить.

Спринт – ітерація з фіксованим терміном, тобто в Scrum повинен бути ритм.

Планування спринту. Перед початком спринту всі збираються і визначають, що потрібно зробити протягом спринту і в якому вигляді.

Огляд спринту або демонстрація: всі збираються і демонструють, що нового в інкременті продукту, дивляться, фіксують зауваження, може бути, міняють найближчі плани.

Ретроспектива: всі збираються і думають, що можна поліпшити в наступному спринті. Наприклад, було багато багів, користувачі незадоволені. Давайте спробуємо в наступному спринті використовувати модульне тестування. Пробуємо, на наступній ретроспективі дивимося, допомогло чи ні, вигадуємо щось ще.

Беклог спринту – верхня частина беклогу. Кількість елементів зазвичай визначається швидкістю команди на заході, який називається «планування спринту», і проводиться перед початком самого етапу спринту. Спринт триває 1-4 тижні (найчастіше 2 тижні). Чим швидше і дешевше можна реліз, тим менше тривалість даного етапу.

Щодня команда проводить 15-хвилинний стендап, *Scrum-мітинг*, на якому всі члени команди синхронізують свої дії. Найчастіше ці зустрічі називають просто «скрам».

Після завершення спринту проводиться демонстрація – «*Огляд*», сенс якої полягає в тому, що команда показує зроблену роботу власнику продукту або комусь ще. Потім проводиться ретроспектива, на якій команда обговорює, що вийшло, а що ні, відбувається розбір робочих процесів, атмосфери в колективі, робляться спроби щось поліпшити. Після цього запускається наступний спринт. У підсумку роботу Scrum-команди можна представити як ланцюжок безлічі спринтів.

Артефакти. Це можуть бути картки на дошці з коротким описом, що собою представляє конкретний функціонал. При цьому зміст картки може являти собою обговорення з власником продукту. Зазвичай це виливається в тікети в трекері, який ви використовуйте.

Беклог продукту – це все тікети, картки або інші вимоги у вигляді елементів беклогу, які є у вашому продукті.

Беклог спринту – найцінніші і пріоритетні вимоги.

6.3.3. Приклади Scrum-практик.

Оцінки.

Agile і Scrum пропонують використовувати таку практику: робити відносні оцінки, тобто «вимірювати в папугах». Це означає, що оцінюється час, який витрачається на вирішення завдання, і скільки він займе папуг. Їх зазвичай називають story point. Ці story point краще не прив'язувати ні до яких тимчасових проміжків. Яким чином робити таку оцінку, чому вона називається відносною? Зазвичай беруть якусь задачу, невелику, стандартну і зрозумілу всім, і оголошують її мірою, еталоном – вона займає 1 папугу, 1 story point. Береться наступні задача, порівнюється з першою – вона виявляється в 2 рази більше. Скільки вона займає? 2 story point. І таким чином ми розкладаємо всі завдання. У результаті ми не оцінюємо кожну задачу за часом, а порівнюємо їх між собою. Якщо десь помиляємося, то це нормально. Головне, щоб у всіх завданнях помилялися однаково. Оцінка робиться командою і в рамках команди.

Що ми можемо зробити після цього? Наприклад, запустити двотижневий спринт і взяти верхні завдання без якихось оцінок. Коли спринт закінчиться, на виході отримаємо інкремент продукту, в який буде включено якусь кількість завдань. Порахуємо, скільки story point ми зробили, і на наступний спринт просто можемо планувати таку ж кількість story point. Подібна оцінка виходить відносною та емпіричною. Шкала оцінок зазвичай підбирається так, щоб розрізняти завдання різних класів (рис. 6.5).

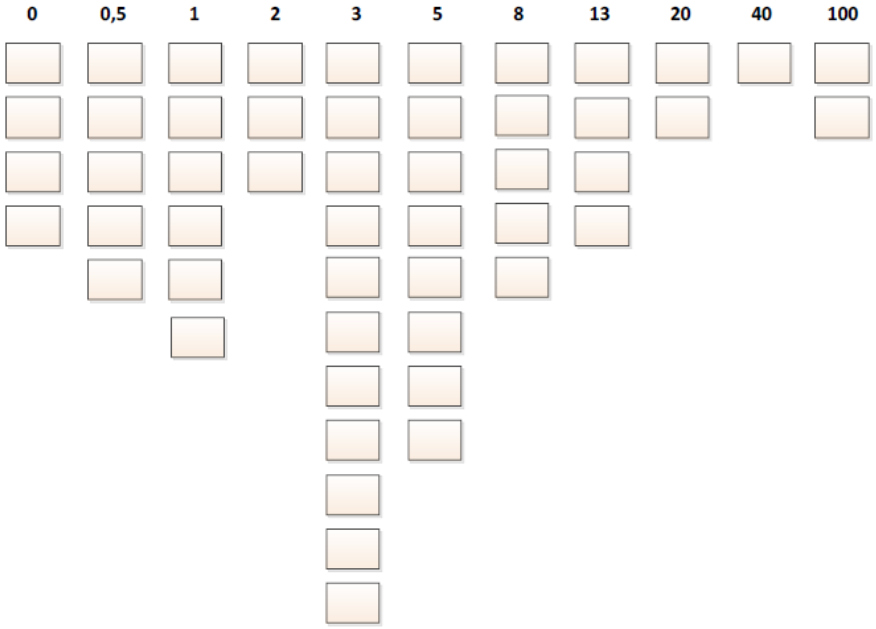


Рисунок 6.5 – Приклад шкали оцінювання

Для формування оцінок зазвичай використовується покер-планування. Це модифікація методу Delphi. Кожному члену команди дається колода карток з числами від 0 до 100. Є ще картка з написом «Я не розумію, що це за завдання» і картка з кавою («Давайте зробимо перерву, я вже замучився займатися цією оцінкою»). Після цього беремо задачу номер такий-то,

читаємо й обговорюємо, що вона собою представляє: «Користувач вводить логін-пароль для того, щоб авторизуватися на сайті». Обговорюємо, як ця авторизація відбувається, де ми зберігаємо користувачів. Потім кожен член команди сорочкою вгору кладе карту з оцінкою, яку вважає потрібною. При цьому різні члени команди один на одного ніяк не впливають, бо зазвичай в команді є тімлідер, провідний, старший розробник, на якого рівняється більшість. Після цього відбувається розтин карт і обговорення (рис. 6.6).

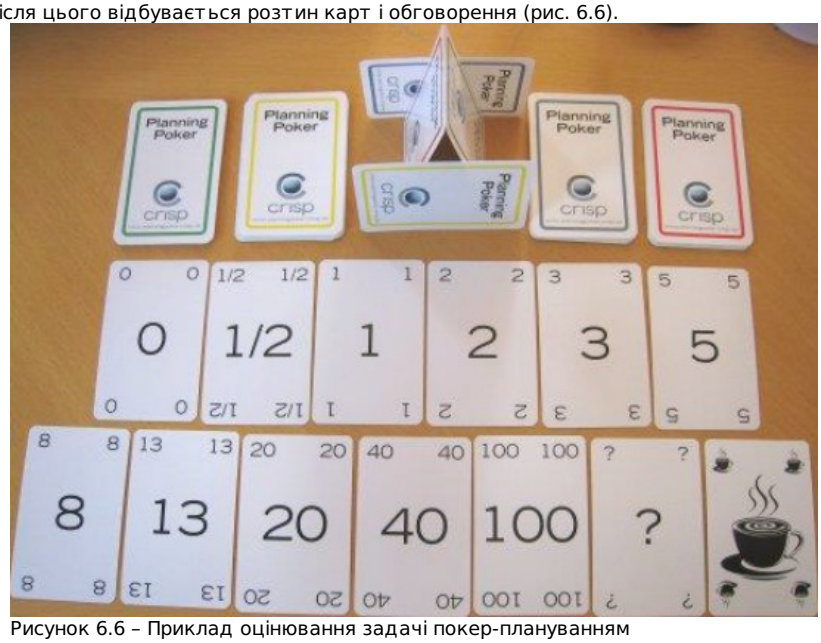


Рисунок 6.6 – Приклад оцінювання задачі покер-плануванням

Згідно з методом Delphi, обговорення відбувається між тими, хто поставив найбільшу і найменшу оцінки. Поставивший найбільшу зазвичай бачить якісь ризики, які не помітили інші члени команди. А людина, що поставила найменшу оцінку, або не розуміє завдання, або бачить спосіб зробити швидше, або вже робила щось подібне. Після цього відбувається другий етап обговорення: знову всі кладуть картки сорочками вгору, потім розкривають їх. Зазвичай оцінки більш-менш згладжуються. Знову проходить етап обговорення, приходять до консенсусу. У результаті записується в трекер, в тікет-систему, або прямо на картку, що у нас це завдання на 3 story point.

Якщо замовник не погоджується відразу на дану систему, то можна її скорегувати: наприклад, якщо ви закінчуєте проект до певної дати, то отримуєте якийсь бонус.

Швидкість команди.

Ми беремо кожен спринт, вимірюємо, скільки story point зробили. І якщо нам треба спланувати наступний спринт, то просто беремо середнє значення з попередніх спринтів (рис. 6.7). Це називається «принцип вчорашньої погоди». Тут важливо те, що ми набираємо найбільш важливі або цінні завдання виходячи з швидкості команди.

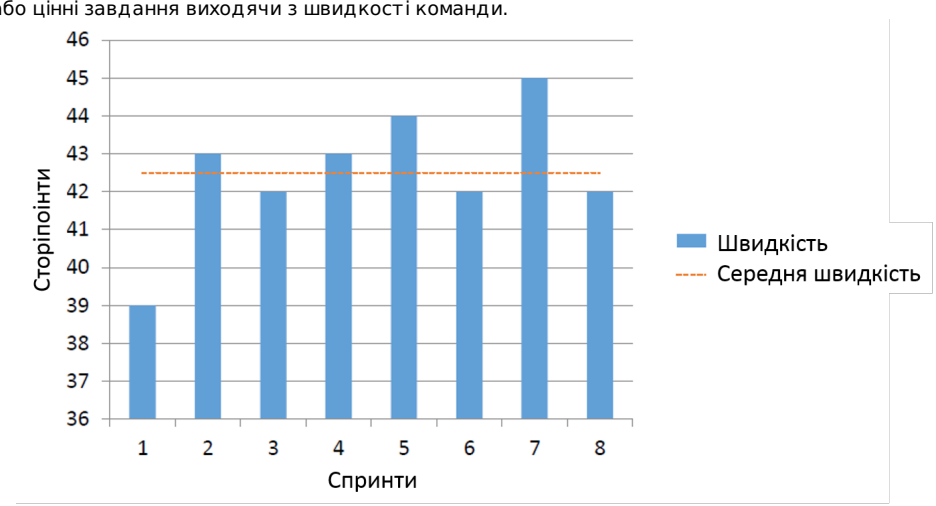


Рисунок 6.7 – Ілюстрація до визначення швидкості спринту

Якщо якась задача не поміщається за розміром, то її можна розбити на більш дрібні, або відмовитися від її вирішення. Треба розуміти, що швидкість не буде рівномірною. Люди працюють зі змінною інтенсивністю, хтось захворіє, хтось працює повільніше через проблем будинку, комусь треба терміново поспілкуватися в соцмережі, хтось взяв відпустку. Завжди потрібно прямо і однозначно говорити власнику продукту: «У нас є завдання A, B, C, ми їх точно встигнемо зробити, вони потрапляють в нашу найнижчу швидкість. Є також завдання D, яке ми, швидше за все, встигнемо зробити. Але є й завдання F, яке може випасти». Така розстановка завдань за мірою важливості дозволяє замовнику для кожного спринту зможе вибирати найважливіші завдання та гарантовано отримувати їх.

Шляховий контроль

Для контролю використовується діаграма згоряння Burn Down Charts (рис. 6.8).

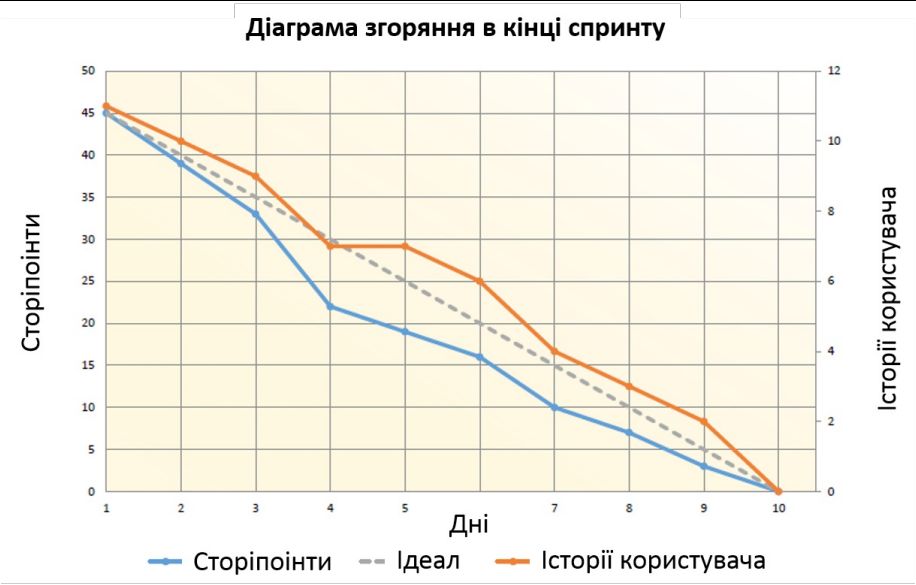


Рисунок 6.8 – Приклад діаграми згоряння

По горизонталі відзначаємо дні – це двотижневий спринт, 10 робочих днів. По вертикалі з одного боку відкладені story point, з іншого – історії користувачів або елементи беклога. Якби ми з вами були роботами, а завдання маленькими і розбитими на шматочки, то ми йшли б по пунктирною лінії – це лінія ідеального Burn Down Charts. Верхня лінія означає скільки ми зробили історій користувачів, нижня – кількість story point. Такі графіки автоматично будуються в багатьох системах для трекінгу завдань.

Якщо ваш графік знаходиться над лінією ідеального Burn Down, то ви відстаєте, повільно працюєте. Тоді до кінця спринту буде не нуль задачок або story point, а десь 20-25. На практиці зазвичай виходить так, як зображено на рис. 6.9.

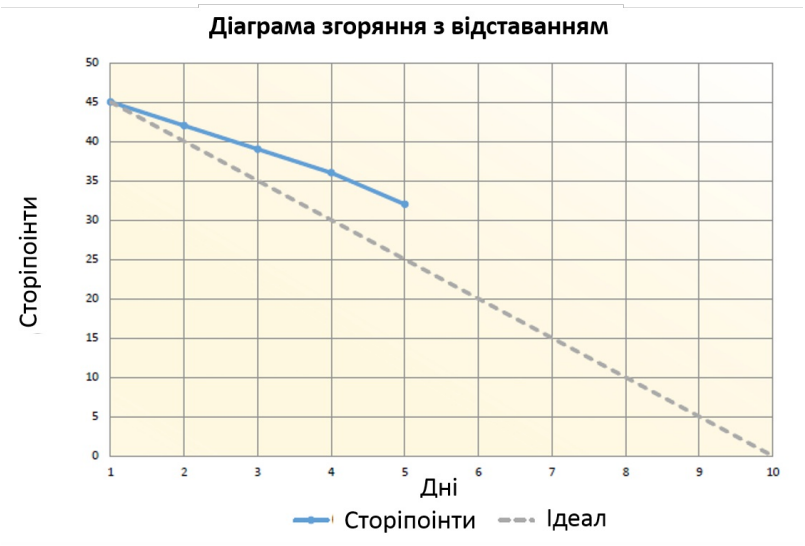


Рисунок 6.9 – Приклад діаграми згоряння з відставанням

Якщо ж Burn Down нижче діагональної лінії (рис. 6.10), це означає, що команда працює з випередженням, тобто, швидше за все, просто взяли мало завдань.

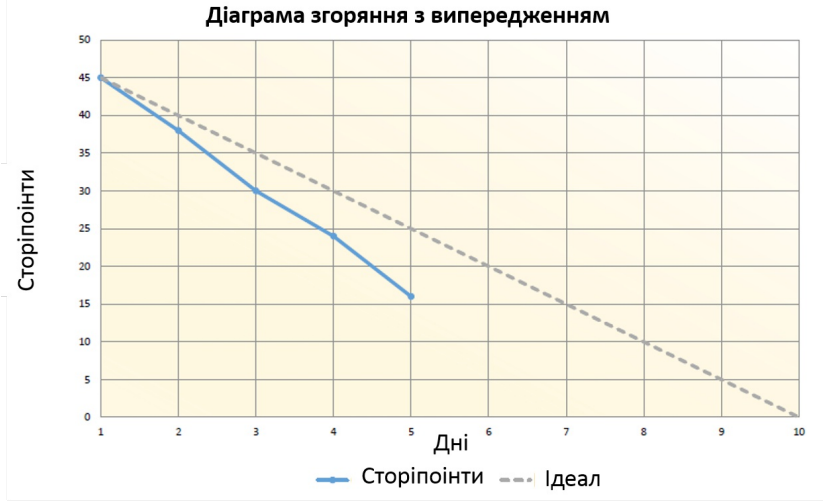


Рисунок 6.10 – Приклад діаграми згоряння з випередженням

Цей графік – артефакт, який можна в кінці двотижневої ітерації проаналізувати і зробити висновки.

Дошки завдань.

Зазвичай в Scrum використовують дошки завдань, хоча вони не є обов'язковим елементом. Команда розподіляє завдання по окремих етапах та розміщує на дошці у вигляді окремих карток. Є електронні реалізації дошок завдань, плагіни для JIRA і т.д. Завдання упорядковуються за ступенем важливості. Коли команда збирається з ранку, оновлюються статуси завдань, їх переносять на інші етапи.

Огляд спринту.

На цій зустрічі в міру можливості беруть участь всі зацікавлені сторони: розробники, користувачі, служба підтримки, системні адміністратори і т. д. Огляд спринту потрібен для того, щоб запустити цикл зворотного зв'язку. Ви показуєте власнику продукту розроблений інкремент. Він дивиться, робить зауваження, вносить пропозиції, віддає їх вам назад в беклог. Ви берете в роботу, створюєте новий інкремент, через один-чотири тижні показуєте і знову отримуєте додаткові зміни у вимогах. Тобто запускаєте цикл, який у результаті приведе вас до продукту, який хоче отримати його власник.

Ретроспектива.

Ретроспектива потрібна для постійних поліпшень. Вся команда збирається і обговорює всі сходинки до самої ретроспективи. Зазвичай це триває від години до чотирьох, може тривати навіть день.

Починається ретроспектива з відкриття. Зазвичай вона займає 5% часу. Завдання – розбуркати всіх присутніх на ретроспективі, тому що дуже часто в командах, особливо айтішних, присутні не дуже товариські люди, але часом мають блискучі думки. Завдання ведучого полягає в тому, щоб розговорити цих людей. Другий етап – збір даних, він займає до половини часу. Команда шукає якісь факти. Їх можна згадати, дістати з трекера, роздрукувати. Після збору фактів починається мозковий штурм: потрібно зрозуміти, в чому проблема, проникнути в суть, згенерувати ідеї, які допоможуть її вирішити. На це йде до третини часу. Зазвичай набирається 5-10 ідей. Далі потрібно втілити ці ідеї в життя. Ми не можемо відразу впровадити code review, розробити якісь інструменти. Тому вибирається максимум одна-дві ідеї, реалізацію яких треба запланувати на наступний спринт. Після цього дякуємо всім і закриваємо ретроспективу. А ще через два тижні можна оцінити, вийшло у нас це чи ні, вивчити інші проблеми.

Є таке поняття, як «Цикл Демінга». Він складається з чотирьох етапів: Plan – Do – Check (Study) – Act (рис. 6.11).

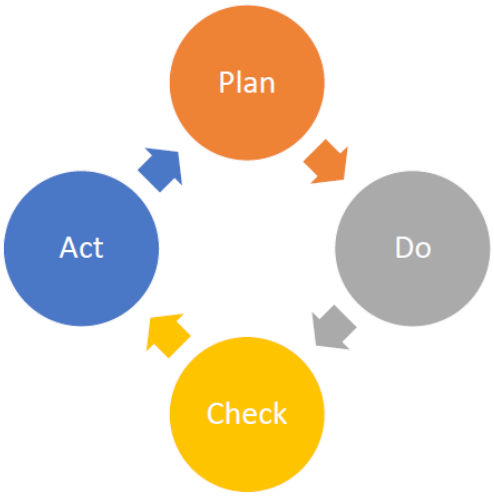


Рисунок 6.11 – Ілюстрація до пояснення циклу Демінга

У ході ретроспективи можна створити план, що ви будете далі змінювати. Скажімо, впроваджуємо unit-тестування – як впроваджуємо, який інструмент використовуємо, яке покриття коду тестами хочемо отримати. Потім настає етап реалізації (це зазвичай спринт, якщо ми говоримо про Scrum), коли ми втілюємо рішення в життя. На наступній ретроспективі можемо перевірити, чи дійсно нам вдалося досягти потрібного ступеня покриття. Можемо подивитися, поменшало чи у нас кількість багів в тих місцях, які ми покрили тестами.

Після цього можемо вносити зміни: наприклад, хотіли зробити покриття 50% – зробили, кількість багів зменшилася, але вони ще залишилися – давайте піднімемо до 70%. Або зробили 70%, цикл прокрутили вдруге, перевіряємо – покращився. Давайте зробимо 90%. Ще раз прокрутили: кількість багів не зменшилася, а витрат на написання і підтримку тестів виходить багато. Давайте зробимо більш слабку границю. Завдяки цьому циклу команда поступово покращує якусь частину процесів. Найпростіший варіант ретроспективи – Real-Time Board Service (рис. 6.12).

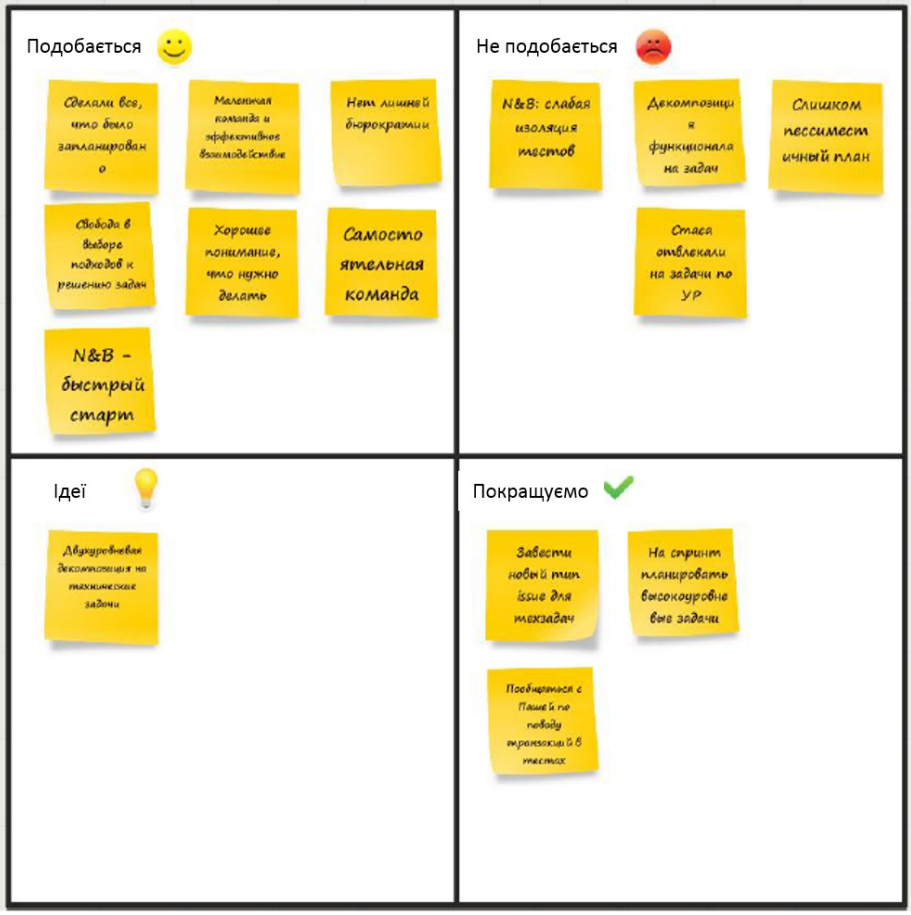


Рисунок 6.12 – Real-Time Board Service

ВИСНОВКИ

Як підсумок можна сказати, що Agile – це серія підходів до розробки програмних продуктів шляхом безперервної і швидкої поставки цінного робочого функціоналу самоорганізованої командою професіоналів у співпраці з замовником. Це не канонічне визначення, а моє власне розуміння Agile.

ЛІТЕРАТУРА

Для підготовки лекції були використані матеріали

1. Мастер-класс Бориса Вольфсона. Основы Agile. Режим доступу.- http://habrahabr.ru/company/mailru/blog/272237/
2. Управление Agile-проектами с YouTrack 4.0. Режим доступу.- http://habrahabr.ru/company/JetBrains/blog/146934/

- 7.1 Основні принципи методології RAD.
- 7.2 Життєвий цикл ПЗ відповідно RAD
  - 7.1.1. Фаза аналізу.
  - 7.1.2. Фаза проектування.
  - 7.1.3. Фаза побудови.
  - 7.1.4. Фаза впровадження.
- 7.3 Оцінка розміру додатків
- 7.4 Логіка додатків, побудованих за допомогою RAD
- 7.4 Застосування RAD.
- Висновки

Одним з можливих підходів до розробки ПЗ в рамках спіральної моделі ЖЦ, що отримала останнім часом широкого поширення, є методологія швидкої розробки додатків RAD (Rapid Application Development).

RAD – це комплекс спеціальних інструментальних засобів швидкої розробки прикладних інформаційних систем, що дозволяють оперувати з певним набором графічних об'єктів, функціонально відображають окремі інформаційні компоненти додатків.

Під цим терміном зазвичай розуміється процес розробки ПЗ, що містить 3 елементи:

- невелику команду програмістів (від 2 до 10 осіб);
- короткий, але ретельно пророблений виробничий графік (від 2 до 6 міс.);
- повторюваний цикл, при якому розробники по мірі набуття форми додатком запитують і реалізують у продукті вимоги, отримані через взаємодію з замовником.

Команда розробників повинна представляти із себе групу професіоналів, які мають досвід в аналізі, проектуванні, генерації коду і тестуванні ПЗ з використанням CASE-засобів. Члени колективу повинні також вміти трансформувати в робочі прототипи пропозиції кінцевих користувачів.

7.1 Основні принципи методології RAD.

Основні принципи методології RAD можна звести до таких:

- Використовується ітераційна (спіральна) модель розробки;
- Повне завершення робіт на кожному з етапів життєвого циклу не обов'язкове;
- В процесі розробки інформаційної системи необхідна тісна взаємодія із замовником і майбутніми користувачами;
- Необхідне використання CASE-засобів і засобів швидкої розробки додатків;
- Необхідне використання засобів управління конфігурацією, що полегшують внесення змін до проекту і супровід готової системи;
- Необхідне використання прототипів, що дозволяє повніше з'ясувати і реалізувати потреби кінцевого користувача;
- Тестування і розвиток проекту здійснюються одночасно з розробкою;
- Розробка ведеться нечисленною і добре керованою командою професіоналів;
- Необхідні грамотне керівництво розробкою системи, чітке планування і контроль виконання робіт.

Засоби RAD дали можливість реалізовувати зовсім іншу в порівнянні з традиційною технологію створення додатків. Інформаційні об'єкти формуються як якісь діючі моделі (прототипи), чиє функціонування узгоджується з користувачем, а потім розробник може переходити безпосередньо до формування закінчених додатків, не втрачаючи з уваги загальної картини проектованої системи.

Можливість використання подібного підходу в значній мірі є результатом застосування принципів об'єктно-орієнтованого проектування, Застосування об'єктно-орієнтованих методів дозволяє подолати одну з головних труднощів, що виникають при розробці складних систем – колосальний розрив між реальним світом (предметною областю описуваної проблеми) і імітаційним середовищем.

Використання об'єктно-орієнтованих методів дозволяє створити опис (модель) предметної області у вигляді сукупності об'єктів – сутностей, поєднуючих дані і методи обробки цих даних (процедури). Кожен об'єкт володіє своєю власною поведінкою і моделює деякий об'єкт реального світу. З цієї точки зору об'єкт є цілком відчутною річчю, яка демонструє певну поведінку.

В об'єктному підході акцент переноситься на конкретні характеристики фізичної або абстрактної системи, що є предметом програмного моделювання. Об'єкти є цілісністю, яка не може бути порушена. Таким чином, властивості, що характеризують об'єкт і його поведінку, залишаються незмінними. Об'єкт може тільки міняти стан, управлятися або ставати в певне відношення до інших об'єктів.

Широку популярність об'єктно-орієнтоване програмування отримало з появою візуальних засобів проектування, коли було забезпечено злиття (інкапсуляція) даних з процедурами, що описують поведінку реальних об'єктів, в об'єкти програм, які можуть бути відображені певним чином в графічному середовищі користувача. Це дозволило приступити до створення програмних систем, максимально схожих на реальні, і досягати найвищого рівня абстракції. У свою чергу, об'єктно-орієнтоване програмування дозволяє створювати більш надійні коди, оскільки у об'єктів програм існує точно визначений і жорстко контрольований інтерфейс.

При розробці додатків за допомогою інструментів RAD використовується множина готових об'єктів, які зберігаються в загальнодоступному сховищі. Однак забезпечується і можливість розробки нових об'єктів. При цьому нові об'єкти можуть розроблятися як на основі існуючих, так і «з нуля».

Інструментальні засоби RAD мають зручний графічний інтерфейс користувача і дозволяють на основі стандартних об'єктів формувати прості програми без написання коду програми. Це є великою перевагою RAD, оскільки в значній мірі скорочує рутинну роботу з розробки інтерфейсів користувача (при використанні звичайних засобів розробка інтерфейсів є досить трудомісткою задачею, що забирає багато часу). Висока швидкість розробки інтерфейсної частини додатків дозволяє швидко створювати прототипи і спрощує взаємодію з кінцевими користувачами.

Таким чином, інструменти RAD дозволяють розробникам сконцентрувати зусилля на сутності реальних ділових процесів підприємства, для якого створюється інформаційна система. У результаті це приводить до підвищення якості розроблюваної системи.

Застосування принципів об'єктно-орієнтованого програмування дозволило створити принципово нові засоби проектування додатків, які були названі засобами візуального програмування. Візуальні інструменти RAD дозволяють створювати складні графічні інтерфейси користувача взагалі без написання коду програми. При цьому розробник може на будь-якому етапі спостерігати, що закладається в основу прийнятих рішень.

Візуальні засоби розробки оперують в першу чергу зі стандартними інтерфейсними об'єктами – вікнами, списками, текстами, які легко можна зв'язати з даними з бази даних і відобразити на екрані монітора. Інша група об'єктів являє собою стандартні елементи управління – кнопки, перемикачі, прапорці, меню і т. п., за допомогою яких здійснюється управління відображеними даними. Всі ці об'єкти можуть бути стандартним чином описані засобами мови, а описи збережені для подальшого повторного використання.

В даний час існує досить багато різних візуальних засобів розробки додатків. Але всі вони можуть бути розділені на дві групи – універсальні і спеціалізовані.

Серед універсальних систем візуального програмування зараз найбільш поширені такі, як C++, Visual Studio. Універсальними ми їх називаємо тому, що вони не орієнтовані на розробку лише додатків баз даних – з їх допомогою можуть бути розроблені додатки майже будь-якого типу, у тому числі й інформаційні програми. Причому програми, які розроблюються за допомогою універсальних систем, можуть взаємодіяти практично з будь-якими системами управління базами даних. Це забезпечується як використанням драйверів ODBC або OLE DB, так і застосуванням спеціалізованих засобів (компонентів).

Спеціалізовані засоби розробки орієнтовані тільки на створення додатків баз даних. Причому, як правило, вони прив'язані до конкретних систем управління базами даних. Як приклад таких систем можна привести Power Builder фірми Sybase (природно, призначений для роботи з СУБД Sybase Anywhere Server) і Visual FoxPro фірми Microsoft.

Оскільки завдання створення прототипів і розробки користувальницького інтерфейсу, по суті, злилися, програміст отримав безперервний зворотний зв'язок з кінцевими користувачами, які можуть не тільки спостерігати за створенням додатків, але й активно брати участь у ньому, коригувати результати і свої вимоги. Це також сприяє скороченню термінів розробки і є важливим психологічним аспектом, який привертає до RAD все більшу кількість користувачів.

7.2 Життєвий цикл ПЗ відповідно RAD

Життєвий цикл ПЗ за методологією RAD складається з чотирьох фаз:

- фаза аналізу і планування вимог;
- фаза проектування;
- фаза побудови;
- фаза впровадження.

7.1.1. Фаза аналізу.

На фазі аналізу і планування вимог користувачі системи визначають функції, які вона повинна виконувати, виділяють найбільш пріоритетні з них, які потребують опрацювання в першу чергу, описують інформаційні потреби. Визначення вимог виконується в основному силами користувачів під керівництвом фахівців-розробників. Обмежується масштаб проекту, визначаються часові рамки для кожної з наступних фаз. Крім того, визначається сама можливість реалізації даного проекту у встановлених рамках фінансування, на даних апаратних засобах і т. п. Результатом даної фази повинні бути список і пріоритетність функцій майбутньої ІС, попередні функціональні та інформаційні моделі ІС.

7.1.2. Фаза проектування.

На фазі проектування частина користувачів бере участь в технічному проектуванні системи під керівництвом фахівців-розробників. CASE-засоби використовуються для швидкого отримання працюючих прототипів додатків. Користувачі, які безпосередньо взаємодіють з ними, уточнюють і доповнюють вимоги до системи, що не були виявлені на попередній фазі. Більш докладно розглядаються процеси системи. Аналізується і, за необхідності, коригується функціональна модель. Кожен процес розглядається детально. За необхідності для кожного елементарного процесу створюється частковий прототип: екран, діалог, звіт, що усуває неясності або неоднозначності. Визначаються вимоги розмежування доступу до даних. На цій же фазі відбувається визначення набору необхідної документації.

Після детального визначення складу процесів оцінюється кількість функціональних елементів розроблюваної системи і приймається рішення про поділ ІС на підсистеми, які піддаються реалізації однією командою розробників за прийнятний для RAD-проектів час – близько 60 – 90 днів. З використанням CASE-засобів проект розподіляється між різними командами (ділиться функціональна модель). Результатом даної фази повинні бути:

- загальна інформаційна модель системи;
- функціональні моделі системи в цілому і підсистем, що реалізуються окремими командами розробників;
- точно визначені за допомогою CASE-засобу інтерфейси між автономно розробленими підсистемами;
- побудовані прототипи екранів, звітів, діалогів.

Всі моделі і прототипи повинні бути отримані із використанням тих CASE-засобів, які будуть використовуватися в подальшому при побудові системи. Дана вимога викликана тим, що в традиційному підході при передачі інформації про проект з етапу на етап може відбутися фактично неконтрольоване спотворення даних. Застосування єдиного середовища зберігання інформації про проект дозволяє уникнути цієї небезпеки.

На відміну від традиційного підходу, при якому використовувалися специфічні засоби прототипування, не призначені для побудови реальних додатків, а прототипи викидалися після того, як виконували завдання усунення неясностей у проекті, в підході RAD кожен прототип розвивається в частину майбутньої системи. Таким чином, на наступну фазу передається більш повна і корисна інформація.

7.1.3. Фаза побудови.

На фазі побудови виконується безпосередньо сама швидка розробка програми. На даній фазі розробники виконують ітеративну побудову реальної системи на основі отриманих в попередній фазі моделей, а також вимог нефункціонального характеру. Програмний код частково формується за допомогою автоматичних генераторів, які отримують інформацію безпосередньо з репозитарію CASE-засобів. Кінцеві користувачі на цій фазі оцінюють одержувані результати і вносять корективи, якщо в процесі розробки система перестає задовольняти визначеним раніше вимогам. Тестування системи здійснюється безпосередньо в процесі розробки.

Після закінчення робіт кожної окремої команди розробників проводиться поступова інтеграція даної частини системи з іншими, формується повний програмний код, виконується тестування спільної роботи даної частини програми з іншими, а потім тестування системи в цілому. Завершується фізичне проектування системи:

- визначається необхідність розподілу даних;
- проводиться аналіз використання даних;
- проводиться фізичне проектування бази даних;
- визначаються вимоги до апаратних ресурсів;
- визначаються способи збільшення продуктивності;
- завершується розробка документації проекту.

Результатом фази є готова система, що задовольняє всім узгодженим вимогам.

7.1.4. Фаза впровадження.

На фазі впровадження проводиться навчання користувачів, організаційні зміни і паралельно з впровадженням нової системи здійснюється робота з існуючою системою (до повного впровадження нової). Так як фаза побудови досить нетривала, планування і підготовка до впровадження повинні починатися заздалегідь, як правило, на етапі проектування системи.

Наведена схема розробки ІС не є абсолютною. Можливі різні варіанти, які залежать, наприклад, від початкових умов, в яких ведеться розробка: розробляється зовсім нова система; вже було проведено обстеження підприємства і існує модель його діяльності; на підприємстві вже існує деяка ІС, яка може бути використана в якості початкового прототипу або повинна

бути інтегрована з розробляється.

7.3 Оцінка розміру додатків

Оцінка розміру додатків виконується на основі так званих функціональних елементів (екрани, повідомлення, звіти, файли і т.п.). Подібна метрика не залежить від мови програмування, на якій ведеться розробка. Розмір програми, яка може бути виконана за методологією RAD, для добре налагодженого середовища розробки ІС з максимальним повторним використанням програмних компонентів, визначається таким чином:

- <1000 функціональних елементів – одна людина;
- 1000-4000 функціональних елементів – одна команда розробників;
- > 4000 функціональних елементів – 4000 функціональних елементів на одну команду розробників.

7.4 Логіка додатків, побудованих за допомогою RAD

Логіка додатка, побудованого за допомогою RAD є подійно-орієнтованою. Це означає наступне: кожен об'єкт, що входить до складу додатка, може генерувати події і реагувати на події, що генеруються іншими об'єктами. Прикладами подій можуть бути: відкриття і закриття вікон, натискання кнопки, натискання клавіші клавіатури, рух миші, зміна даних в базі даних і т. п.

Розробник реалізує логіку програми шляхом визначення обробника кожної події – процедури, виконуваної об'єктом при настанні відповідної події. Наприклад, обробник події «натискання кнопки» може відкрити діалогове вікно. Таким чином, управління об'єктами здійснюється за допомогою подій.

Обробники подій, пов'язаних з управлінням базою даних (DELETE, INSERT, UPDATE), можуть реалізовуватися у вигляді тригерів на клієнтському або серверному вузлі. Такі обробники дозволяють забезпечити посилальну цілісність бази даних при операціях видалення, вставки і оновлення, а також автоматичну генерацію первинних ключів.

7.4 Застосування RAD.

Застосування технології RAD доцільно, коли:

- потрібне виконання проекту у стислі терміни (90 днів). Швидке виконання проекту дозволяє створити систему, що відповідає вимогам сьогоднішнього дня. Якщо система проектується довго, то вельми висока ймовірність, що за цей час істотно зміняться фундаментальні положення, що регламентують діяльність організації, тобто, система морально застаріє ще до завершення її проектування
- нечітко визначені вимоги до ПЗ. У більшості випадків Замовник дуже приблизно уявляє собі роботу майбутнього програмного продукту і не може чітко сформулювати всі вимоги до ПЗ. Вимоги можуть бути взагалі не визначені до початку проекту або можуть змінюватися по ходу його виконання
- проект виконується в умовах обмеженості бюджету. Розробка ведеться невеликими RAD групами у короткі терміни, що забезпечує мінімум трудовитрат і дозволяє вписатися у бюджетні обмеження
- інтерфейс користувача (GUI) є головним фактором. Немає сенсу примушувати користувача малювати картинки. RAD технологія дає можливість продемонструвати інтерфейс в прототипі, причому досить скоро після початку проекту
- проект великий, але піддається поділу на більш дрібні функціональні компоненти. Якщо передбачувана система велика, необхідно, щоб її можна було розбити на дрібні частини, кожна з яких має чітку функціональність. Вони можуть випускатися послідовно або паралельно (в останньому випадку залучається кілька RAD груп)
- ПЗ не володіє великою обчислювальною складністю

Слід, однак, відзначити, що методологія RAD, як і будь-яка інша, не може претендувати на універсальність, вона хороша в першу чергу для відносно невеликих проектів, що розробляються для конкретного замовника. Якщо ж розробляється типова система, яка не є закінченим продуктом, а являє собою комплекс типових компонент, централізовано супроводжуваних, які адаптуються до програмно-технічних платформ, СУБД, засобів телекомунікації, організаційно-економічних особливостей об'єктів впровадження і інтегрованих з існуючими розробками, на перший план виступають такі показники проекту, як керованість і якість, які можуть увійти в суперечність з простотою і швидкістю розробки. Для таких проектів необхідні високий рівень планування і жорстка дисципліна проектування, суворе дотримання заздалегідь розробленим протоколам і інтерфейсам, що знижує швидкість розробки.

Методологія RAD непридатна для побудови складних розрахункових програм, операційних систем або програм управління космічними кораблями, тобто програм, що вимагають написання великого обсягу (сотні тисяч рядків) унікального коду.

Не підходять для розробки за методологією RAD додатки, в яких відсутня яскраво виражена інтерфейсна частина, наочно визначає логіку роботи системи (наприклад, додатки реального часу) і додатки, від яких залежить безпека людей (наприклад, керування літаком або атомною електростанцією), так як ітеративний підхід передбачає, що перші кілька версій напевно не будуть повністю працездатні, що в даному випадку виключається.

Висновки

Застосування методології RAD найбільш ефективно при створенні порівняно невеликих систем, що розробляються для конкретного замовника.

Як підсумок перерахуємо основні принципи методології RAD:

- розробка додатків ітераціями;
- необов'язковість повного завершення робіт на кожному з етапів життєвого циклу;
- обов'язкове залучення користувачів до процесу розробки ІС;
- необхідне застосування CASE-засобів, що забезпечують цілісність проекту;
- застосування засобів управління конфігурацією, що полегшують внесення змін до проекту і супровід готової системи;
- необхідне використання генераторів коду;
- використання прототипування, що дозволяє повніше з'ясувати і задовольнити потреби

кінцевого користувача;
- тестування і розвиток проекту, здійснювані одночасно з розробкою;
- ведення розробки нечисленної добре керованої командою професіоналів;
- грамотне керівництво розробкою системи, чітке планування і контроль виконання робіт.

RAD припускає, що розробка програмного забезпечення (ПЗ) здійснюється невеликою командою розробників за термін близько трьох-чотирьох місяців шляхом використання інкрементного прототипування із застосуванням інструментальних засобів візуального моделювання та розробки. Технологія RAD передбачає активне залучення Замовника вже на ранніх стадіях – обстеження організації, вироблення вимог до системи. Причини популярності RAD випливають з переваг, які забезпечує ця технологія.

Найбільш істотними з них є:

- Висока швидкість розробки;
- Низька вартість;
- Висока якість.

Остання із зазначених властивостей має на увазі повне виконання вимог Замовника – як функціональних, так і нефункціональних, з урахуванням їх можливих змін у період розробки системи, а також отримання якісної документації, що забезпечує зручність експлуатації та супроводу системи. Це означає, що додаткові витрати на супровід відразу після поставки будуть значно меншими. Таким чином, повний час від початку розробки до отримання прийнятного продукту при використанні цього методу значно скорочується.

- 7.1 Права і ролі в екстремальному програмуванні
  - 7.1.1. Замовник.
  - 7.1.2. Розробник.
  - 7.1.3. Ролі всередині ролі.
    - Сторона замовника
    - Сторона розробника
  - 7.1.4. Зовнішні ролі.
- 7.2 Основні цінності XP
  - 7.2.1. Спілкування.
  - 7.2.2. Простота.
  - 7.2.3. Зворотній зв'язок.
  - 7.2.4. Сміливість, кураж.
  - 7.2.5. Повага (нова цінність).
- 7.3 Принципи XP
  - 7.3.1. Людяність.
  - 7.3.2. Економіка.
  - 7.3.3. Взаємна вигода.
  - 7.3.4. Подібність.
  - 7.3.5. Постійний розвиток.
  - 7.3.6. Різноманітність.
  - 7.3.7. Міркування.
  - 7.3.8. Потік.
  - 7.3.9. Нові можливості.
  - 7.3.10. Надмірність.
  - 7.3.11. Невдачі.
  - 7.3.12. Відповідальність.
- 7.4 Практики XP
  - 7.4.1.Основні практики.
    - Аналіз вимог і планування
    - Команда і людський фактор
    - Проектування
    - Програмування та випуск продукту
  - 7.4.2. Додаткові практики.
    - Аналіз вимог і планування
    - Команда і людський фактор
    - Проектування
    - Програмування та випуск продукту
- Висновки

Екстремальне програмування – методологія швидкої розробки програмного забезпечення. Складається з набору методик і принципів, що дозволяють як окремо, так і в комплексі, оптимізувати процес розробки. Цей підхід також регламентує права розробників і замовників.

Автори методології – Кент Бек, Уорд Каннінгем, Мартін Фаулер та інші.

Назва методології виходить з ідеї застосувати корисні традиційні методи і практики розробки програмного забезпечення, піднявши їх на новий «екстремальний» рівень. Так, наприклад, практика виконання ревізії коду, що містить у перевірці одним програмістом коду, написаного іншим програмістом, в «екстремальному» варіанті являє собою «парне програмування», коли один програміст займається кодуванням, а його напарник в цей же час безперервно переглядає щойно написаний код.

7.1 ПРАВА І РОЛІ В ЕКСТРЕМАЛЬНОМУ ПРОГРАМУВАННІ

В екстремальному програмуванні чітко описані всі ролі. Кожна роль передбачає характерний набір прав і обов'язків. Тут існують дві ключові ролі: замовник і розробник.

7.1.1. Замовник.

Замовник – це людина або група людей, зацікавлених у створенні конкретного програмного продукту. Він має такі права та обов'язки:

- зафіксувати терміни випуску версій продукту;
- приймати рішення щодо запланованих складових програми;
- знати орієнтовну вартість кожної функціональної складової;
- приймати важливі бізнес-рішення;

- знати поточний стан системи;
- змінювати вимоги до системи, коли це дійсно важливо.

Для успішного використання своїх прав замовник повинен покладатися на дані, що надаються розробниками.

7.1.2. Розробник.

Розробник – один або група від двох до десяти чоловік, які займаються безпосередньо програмуванням і супутніми інженерними питаннями. Розробник наділений такими правами і обов'язками:

- отримати достатнє знання питань, які повинні бути запрограмовані;
- мати можливість з'ясування деталі в процесі розробки;
- надавати орієнтовні, але відверті оцінки трудовитрат на кожну функціональну частину або історію користувача;
- коригувати оцінки на користь більш точних в процесі розробки;
- надавати оцінку ризиків, пов'язаних з використанням конкретних технологій.

7.1.3. Ролі всередині ролі.

Кожна з базових ролей екстремального програмування може бути уточнена більш дрібними ролями. В XP дозволено суміщення ролей в рамках однієї людини.

Сторона замовника

Укладач історій – фахівець предметної області, володіє здібностями доступно викласти і описати вимоги до розроблюваної системі. Ця людина або група людей відповідальні за написання історій користувача і прояснення непорозуміння з боку програмістів.

Приймальник – людина, яка контролює правильність функціонування системи. Добре володіє предметною областю. В обов'язки входить написання приймальних тестів.

Великий бос – стежить за роботою всіх ланок, від розробників до кінцевих користувачів. Він контролює впровадження системи і супутні організаційні моменти. Може бути також інвестором проекту.

Сторона розробника

Програміст – людина, що займається кодуванням і проектуванням на низькому рівні. Він достатньо компетентний для вирішення поточних завдань розробки та надання правдивих оцінок запланованим завданням.

Інструктор – досвідчений розробник, що добре володіє всім процесом розробки і його методиками. Несе відповідальність за навчання команди аспектам процесу розробки. Впроваджує та контролює правильність виконання методик процесу, який використовується. Звертає увагу команди на важливі, але з якихось причин втрачені моменти розробки. Разом з тим інструктор, як і будь-яка інша людина з увагою ставиться до ідей інших членів команди.

Спостерігач – член команди розробників, що користується довірою всієї групи, який стежить за прогресом розробки. Він порівнює попередні оцінки трудовитрат і реально витрачені, виводячи кількісні показники роботи команди. Це такі як середня швидкість і процентне співвідношення виконаних і запланованих завдань. Дана інформація надається замовнику для своєчасного контролю над ситуацією. Частина цієї інформації ненав'язливо надається і розробникам, для знання стану проекту, місць виникнення ускладнень і що ще можна встигнути зробити.

Дипломат – комунікабельна особистість, яка ініціює спілкування між членами команди. Так як документообіг мінімізований, важливо постійне спілкування і передача досвіду всередині команди, краще розуміння вимог до системи. Дипломат регулює і спрощує спілкування між замовниками та розробниками. Є важливою ланкою на зборах. Він перешкоджає недомовкам, розпалу пристрастей і непотрібним сварок. Дипломат не може нав'язувати своєї думки присутнім на зборах.

7.1.4. Зовнішні ролі.

Консультант – фахівець, що володіє конкретними технічними навичками, для допомоги розробникам в складних завданнях. Зазвичай залучається з боку.

7.2 ОСНОВНІ ЦІННОСТІ XP

Нова версія екстремального програмування базується вже на п'яти цінностях (замість чотирьох на початку створення методології), які, як вважає Бек, є головними складовими успіху будь-якого проекту з розробки програмного забезпечення. Основні цінності – це не тільки керівництво з розробки ПЗ, це ще й джерело натхнення всій методології.

7.2.1. Спілкування.

Більша частина проблем і помилок завжди пов'язана з браком спілкування. Отже, необхідно максимально збільшити спілкування між членами команди програмістів, а також між програмістами і замовниками. Найефективнішим є пряме спілкування між людьми. Не можна також забувати і про інший вид спілкування - між артефактами і людьми, які їх читають. Всі артефакти повинні нести адекватну, нестаріючий інформацію. Крім того, їх має бути зручно читати.

7.2.2. Простота.

Простота – найбільш раціональна з усіх цінностей XP. Полягає вона в наступному: «Зупиняйся на найпростішому рішенні, яке дозволяє виконати завдання». Однак просте рішення (саме просте, а не примітивне) якраз найважче знайти. Чим простіше рішення, тим більше за ним стоїть досвіду, ідей і важкої праці. Такі рішення вимагають спілкування, для них потрібно набагато менше коду, вони покращують якість програмного додатку. Основна вимога звучить як «не варто

заздалегідь планувати повторне використання одного і того ж рішення; чим простіше система, тим легше додавати в неї функціональність по мірі необхідності ».

7.2.3. Зворотній зв'язок.

У вас завжди повинна бути можливість визначити, наскільки система, яка проектується, далека від необхідного набору функціональності. Кращі інструменти зворотного зв'язку – це безпосереднє спілкування з замовником і набір автоматизованих тестів, який зростає разом з системою. З одного боку, зворотний зв'язок є важливою складовою спілкування: щоб дізнатися думку замовника, вам треба з ним спілкуватися. З іншого, отримані таким чином відомості стають темою для подальшого спілкування. Зворотній зв'язок допомагає знайти просте рішення. Найчастіше прості рішення досягаються методом проб і помилок. І знову-таки, чим простіше система, тим легше підтримувати зворотний зв'язок.

7.2.4. Сміливість, кураж.

Всі існуючі методології та процеси розробки призначені для того, щоб приборкати і зменшити наш страх. Чим більше страху ми відчуваємо перед яким-небудь проектом, тим серйозніше і «важче» має бути методологія. Спілкування, простота і зворотний зв'язок дають нам можливість сміливо приступати навіть до серйозного рефакторингу системи або до великих змін у вимогах. Сміливість і кураж самі по собі досить небезпечні, але в сукупності з іншими цінностями – це найпотужніший інструмент для здійснення великих змін в системі.

7.2.5. Повага (нова цінність).

Всі чотири попередні цінності мають на увазі повагу членів команди один до одного. Якщо програмісти не поважають один одного і свою роботу, то жодна методологія їм не допоможе. Ви повинні проявляти повагу до колег по роботі, їх праці, до вашої компанії, а також до тих, в чиє життя увійде написаний вами додаток.

Як бачите, в основних цінностях екстремального програмування немає жодних порад щодо того, як керувати проектом або як писати програмний код. Це описується в практиках XP, але перш, ніж перейти до практик, ми повинні зупинитися на принципах.

7.3 ПРИНЦИПИ XP

У новій версії XP принципи є ніби проміжною ланкою між вельми синтетичними й абстрактними цінностями і практиками, в яких даються конкретні вказівки по розробці ПЗ. Тепер в XP розрізняють такі чотирнадцять принципів:

7.3.1. Людяність.

Програмне забезпечення створюється людьми, тому людський фактор залишається основним ключем до розробки якісного програмного продукту. Екстремальне програмування ставить своєю метою вигоду як окремих людей, так і цілих організацій, щоб користь від використання методології відчувалася обома сторонами. Звичайно ж, тут потрібний баланс: якщо ми переоцінимо потреби колективу розробників, це може призвести до зниження продуктивності праці, люди не будуть працювати належним чином, що приведе компанію до збитків. А збитки, в свою чергу, можуть призвести до закриття проекту або навіть всієї фірми, що боляче вдарить по людям, які в ній працювали. Якщо ж ви переоціните потреби організації, люди почнуть перероблять, між ними почастішають конфлікти. Це теж приведе компанію до втрат.

Автор книги про екстремальне програмування Кент наводить такий список потреб команди:

- Базова безпека – необхідність справлятися з роботою;
- Досягнення – почуття власної корисності на роботі;
- Належність – здатність зараховувати себе до групи;
- Зріст – можливість розширювати і поглиблювати свої навички, бачити нові перспективи;
- Близькість – здатність розуміти інших і бути зрозумілим.

7.3.2. Економіка.

ПЗ повинно володіти певною цінністю (business value). У XP є два ключових економічних моменти: цінність програмного продукту на поточний момент і цінність додаткових можливостей (value of options). Згідно з першою, долар, зароблений сьогодні, коштує більше, ніж долар зароблений завтра, тому чим швидше ми випускаємо програмний продукт і починаємо заробляти гроші, тим більший прибуток. Це прямо пов'язано з ціною додаткових можливостей програми. Так, якщо ви можете відкласти архітектурні зміни до того моменту, коли їх необхідність стане абсолютно очевидною, то збережете своєї компанії великі гроші. Практики XP вітають інкрементальний дизайн, увагу до того, що приносить вигоду клієнту, і відношення до розробки, яке можна було б охарактеризувати словами «плати за те, чим користуєшся». Все це значно спрощує процес прийняття рішень.

7.3.3. Взаємна вигода.

Всяка діяльність повинна приносити вигоду задіяним людям та організаціям. Це, мабуть, сам важливий з принципів XP, хоча його дуже складно дотримуватися. З будь-якої проблеми легко знайти вихід, при якому постраждає одна з взаємодіючих сторін. І нерідко нам дуже хочеться йти саме таким шляхом. Однак подібні рішення завжди погіршують становище, оскільки вони псують відносини і руйнують робоче середовище. Щоб не збільшувати кількість проблем, вам знадобляться навички, які забезпечать вигідну співпрацю і вам, і вашим клієнтам, причому і зараз, і в майбутньому.

7.3.4. Подібність.

У природі часто зустрічаються фрактальні структури, дуже схожі між собою, але існуючі на різних рівнях. Точно такі ж принципи можна застосувати і до розробки програмного забезпечення: ми можемо використовувати одні й ті ж ідеї для вирішення проблем, що

виникають в різних контекстах. Наприклад, одна з основних складових методології XP полягає в тому, щоб спочатку написати тести, які свідомо не будуть проходити, а вже потім – написати код, після якого тести пройдуть успішно. Те ж саме можна «масштабувати» на різні рівні тимчасової шкали: впродовж цілого кварталу ви складаєте перелік завдань, які має вирішувати додаток, а потім короткі «розповіді», які описують їх більш докладно. На початку тижня ви вибираєте ті «розповіді», де описані завдання, над якими ви будете працювати протягом цього тижня, а вже потім пишете тести приймання (acceptance tests) і, врешті-решт, приступаєте до програмного коду, завдяки якому ці тести будуть працювати. Якщо звузити часовий проміжок до декількох годин – ви спочатку пишете unit тести, а потім код, який забезпечує їх виконання.

7.3.5. Постійний розвиток.

Постійний розвиток і рух вперед – ключ до розуміння XP. Звичайно, досконалість недосяжна, проте треба постійно до нього прагнути. Кожен день треба намагатися працювати якомога краще і придумувати нові способи зробити роботу ще більш ефективною. Таким чином, з кожною ітерацією продукт стає все краще і краще – як з точки зору якості коду, так і з точки зору функціональних можливостей. Це забезпечується за рахунок відгуків замовника, автоматичних тестів, і звичайно, самої команди розробників.

7.3.6. Різноманітність.

Якщо всі члени команди схожі один на одного, робота в ній може виявитися дуже комфортною, але малоефективною. Краще, коли в команді є представники з різних областей знань, різні характери. Це дозволяє краще знаходити і вирішувати проблеми. Звичайно, в такій команді можуть виникати конфлікти, які доведеться якось вирішувати. Однак якщо ви здатні вирішувати конфліктні ситуації і визначати найкращу з альтернативних думок, слід віддавати перевагу «неоднорідним» командам. Вони вміють знаходити несподівані рішення в складних ситуаціях, що в нашій області дуже важливо.

7.3.7. Міркування.

Ефективно працюючі програмісти не просто пишуть код. Вони запитують себе, як вони працюють і чому вони роблять дану систему саме так, а не інакше. Людям потрібно чітко бачити причини, що стоять за успіхом (або провалом) їхнього продукту. Не треба ховати помилки. Куди корисніше відкрито визнати їх і спробувати винести з них корисний урок на майбутнє. Під час щоквартальних та щотижневих ітерацій відведіть деякий час на обговорення того, як рухається проект, і які поліпшення можна було б внести в процес розробки. Але не треба надто вже загострювати на цьому увагу. У нашій індустрії є багато прикладів того, як програмісти настільки переключалися на вдосконалення процесу роботи, що на саму роботу у них не залишалося часу! Спочатку йде робота – потім міркування – потім знову робота.

7.3.8. Потік.

Потік (flow): в даному випадку, це означає постійну розмірену розробку якісного програмного забезпечення, що включає в себе всі відповідні види діяльності. У методології XP прийнято вважати, що всі види діяльності з розробки ПЗ повинні протікати безперервно, подібно до потоку. Цим вона відрізняється від інших методологій, де розробка розпадається на послідовність певних фаз, останньою з яких є випуск програмного продукту. Тільки завдяки безперервному потоку розробки програмісти можуть раніше дізнатися думку замовників про систему, отримати впевненість, що робота ведеться в належному напрямку, а крім того, уникнути труднощів останньої інтеграції перед випуском продукту.

7.3.9. Нові можливості.

В проблемах треба бачити передусім можливість щось поліпшити. Проблеми неминучі, але щоб досягти досконалості мало просто «вирішити проблему». Треба перетворити проблему в можливість дізнатися щось нове, знайти найкраще рішення.

7.3.10. Надмірність.

Дійсно складні або критичні для проекту проблеми треба вирішувати кількома способами одночасно. У цьому випадку, якщо одне рішення виявиться неспроможним, інші можуть допомогти запобігти катастрофі. У таких випадках в підсумку надмірність з лишком окупається. Проблеми, пов’язані з функціональністю програмного забезпечення, потрібно шукати і вирішувати різними способами: парним програмуванням, автоматизованими тестами, роботою в загальному приміщенні, активним залученням замовника, і т.д. Зрозуміло, в цьому є деяка надмірність – багато недоліків будуть виявлятися по кілька разів. Проте якість продукту – найголовніша мета – все окупить. Втім, якщо з допомогою якоїсь практики вдається виявити тільки вже відомі дефекти, її треба скасовувати через непотрібність.

7.3.11. Невдачі.

Невдачі: якщо щось не виходить, не бійтеся невдач. Не знаєте, як краще написати частину нової функціональності? Спробуйте зробити це трьома-чотирма різними способами. Навіть якщо всі виявляться невдалими, ви багато чому навчитеся. Чи корисні невдачі? Так, якщо ми виносимо з них цінні уроки. Куди гірше відкладати щось до останнього моменту, намагаючись знайти єдино вірне рішення.

7.3.12. Якість

Якість завжди має бути на висоті. Випускати продукт низької якості, щоб заощадити кошти або час, – свідомо неправильне рішення. Якраз навпаки, робота над якістю системи сприяє поліпшенню інших його властивостей, наприклад, продуктивності та ефективності. Проте, якщо ви відкладаєте активну фазу розробки, в надії знайти ідеальне рішення, ви не будете просуватися вперед. Набагато ефективніше спробувати якийсь варіант, з’ясувати, в чому його недоліки, і швидко їх виправити.

7.3.13. Маленькими кроками

Якщо довго готувати серйозні великі зміни, а потім внести їх у систему «одним махом», весь проект може опинитися під загрозою зриву. Набагато надійніше просуватися вперед

маленькими кроками – нехай кроки ці невеликі, зате ви можете бути впевнені, що проект рухається в потрібному напрямку. Крім того, маленькими кроками можна рухатися досить швидко: за короткий час команда програмістів може зробити дуже багато таких «кроків», при цьому постійно отримувати відгуки і коригувати просування робіт. Ще один плюс невеликих змін полягає в тому, що невелика зміна, як правило, може призвести лише до невеликих проблем. А ось чим більше крок і чим серйозніше зміни, тим більший потенційний шкода може він принести всьому проекту.

7.3.12. Відповідальність.

Відповідальність можна взяти на себе тільки в добровільному порядку. Звичайно, будь-який начальник може наказати програмісту: «Роби те, роби це», але такий підхід не працює. Ви неминуче будете вимагати або набагато менше того, що потрібно, або набагато більше того, що може даний програміст. Тому отримуючи вказівки, людина сама повинна прийняти рішення: чи бере вона на себе відповідальність, або ж нехай цією проблемою займається хтось інший.

7.4 ПРАКТИКИ XP

Нова версія екстремального програмування налічує тридцять основних практики та одинадцять додаткових. Спочатку потрібно застосувати основні практики, причому кожна з них може відповідним чином поліпшити процес розробки. Тільки після цього можна приступати до додаткових практик, які вимагають досвіду роботи з основними практиками, і практично не застосовуються без них.

У цілому ж можна сказати, що всі 24 практики дуже важливі для процесу розробки, і повинні бути застосовані повністю. Тільки так проект отримає вигоду від екстремального програмування.

Доцільним є розподіл практик на чотири категорії:

- Аналіз вимог і планування;
- Команда і людський фактор;
- Проектування;
- Програмування та випуск продукту.

7.4.1.Основні практики.

Аналіз вимог і планування

«Оповідання»: функціональність програми описується короткими «оповіданнями», в яких робота системи викладена з погляду замовника. Ці «розповіді» також є основною рушійною силою розробки програми.

Щотижневий цикл: вся розробка проекту відбувається у вигляді низки щотижневих циклів. На початку тижня відбувається зібрання, на якому замовник вибирає, які «розповіді» треба зробити за цей тиждень.

Щоквартальний цикл: планування в більшому масштабі відбувається кожен квартал. Воно складається з обговорень роботи команди і темпів розробки.

Слабина: уникайте обіцянок, які не зможете виконати. У будь-який план включайте завдання, які ви зможете викинути, якщо не будете укладатися в термін. У цьому випадку у вас буде вихід навіть у разі непередбачених проблем.

Команда і людський фактор

Робота в одному приміщенні: команда розробників повинна сидіти в одному великому приміщенні – це полегшує спілкування.

Команда як єдине ціле: команда повинна складатися з людей, що володіють всіма необхідними для проекту навичками і знаннями. Всіх їх повинно об'єднувати почуття приналежності спільній справі, вони повинні всіляко підтримувати один одного.

Інформативність оточення: в робочих приміщеннях повинні бути інформативні постери та інші наочні посібники, які показували б статус проекту та іншу інформацію про виконану роботу.

Енергійна робота: люди не повинні бути виснажені роботою, їм треба зберігати свіжість і енергійність, щоб фокусуватися на завданнях і вміти ефективно їх вирішувати. Отже, треба жорстко обмежити понаднормову роботу, так щоб у кожного залишався час і на особисте життя. У колишньої версії методології це називалося «*прийнятний темп розробки*».

Парне програмування: код завжди пишуть два програмісти, що сидять за одним комп'ютером.

Проектування

Інкрементне проектування: згідно XP, не слід займатися докладним проектуванням системи на самому початку робіт. Замість цього команда розробників прагне якнайскоріше почати писати програмний код, щоб отримати відгуки користувачів про систему, і покращувати її по ходу справи. Звичайно, щоб написати хороший код, система повинна бути належним чином спроектована. Цього XP не заперечую. Питання лише – коли займатися проектуванням. Згідно екстремального програмування, проектування має відбуватися інкрементально під час написання програмного коду. Особливо корисно робити це, щоб прибирати непотрібне дублювання.

Спочатку тести: перед тим, як редагувати старий код або писати новий, потрібно написати тести, які будуть його перевіряти. Це допоможе вирішити чотири проблеми:

- «Програмування по-ковбойські»: під час написання коду так легко захопитися і почати писати код для всіх завдань підряд, які приходять на розум. Якщо ж спочатку написати тести, які потім будуть перевіряти код, нам волею-неволею доведеться зосередитися на завданні, яке ми намагаємося вирішити, а також перевірити, наскільки правильно ми спроектували дану частину системи.
- Злагодженість і єдність: якщо написати тест важко, значить, у вас проблеми з дизайном системи, а не з тестуванням або програмуванням. Коли програмний код розбитий на функціонально пов'язані модулі з мінімальною кількістю двосторонніх залежностей між

ними, тестувати його не складе великого труда.
- Довіра: якщо ви пишете код, який працює, і документуєте його за допомогою автоматизованих тестів, ваші колеги будуть довіряти вам.
- Ритм: під час програмування можна легко захопитися і блукати в коді протягом декількох годин. Якщо ви привчите себе до ритму «тест, код, рефакторинг, тест, код, рефакторинг», цього ніколи не трапиться.

Програмування та випуск продукту

Десятихвилинна збірка: повинна бути можливість забрати систему (з урахуванням прогону всіх тестів) за десять хвилин. Це дозволить часто повторювати операцію і отримувати відгуки про розроблюваний продукт.

Постійна інтеграція: розробники повинні викладати в репозитарій результати своєї роботи кожні 2 години, щоб уникнути проблем з інтеграцією нового коду.

7.4.2. Додаткові практики.

Аналіз вимог і планування

Безпосереднє залучення замовника: люди, для яких ви пишете програмний продукт, повинні стати частиною команди і вносити свій вклад в щоквартальне і щотижневе планування.

Інкрементна поставка продукту: коли вам належить цілком змінити існуючу систему, починайте процес зі зміни декількох функцій, і так поступово замініть всю систему. Уникайте підходу, який можна виразити словами «Все або нічого!»

Контракт з обумовленим обсягом робіт: контракт на розробку програмного забезпечення включає в себе час, витрати і якість системи, проте точні обсяги цієї системи треба обмовляти в процесі роботи. Уклавши з замовником серію невеликих контрактів, можна значно знизити ризики.

Плата за використання: зазвичай замовник платить за кожен випуск програмного продукту. Це нерідко дає привід для конфліктів між розробниками і замовником, який хоче внести якомога більше нової функціональності в найменшу кількість випусків продукту. Якщо обчислювати грошима безпосередню роботу над функціональністю, замовник буде одержувати більш точну та своєчасну інформацію, і зможе точніше направляти розробку продукту.

Команда і людський фактор

Стабільність: команда розробників повинна працювати в одному і тому ж складі впродовж кількох проектів. Ті зв'язки, які виникають між людьми, воістину безцінні, тому намагайтеся перерозподіляти людей якомога рідше.

«*Усушка і утруска*»: якщо команда стає все більш продуктивною, не слід збільшувати її навантаження. Залиште обсяг робіт колишнім, але виділіть вільних членів цієї команди з тим, щоб вони створювали свої власні нові команди.

Проектування

Аналіз причин і наслідків: кожен раз, коли ви знаходите помилку в системі, виправляйте не тільки її, але і її причину. В іншому випадку ця помилка може повторитися в майбутньому.

Програмування та випуск продукту

Код і тести: тільки програмний код і тести є постійними артефактами системи, які треба зберігати. Все інше може бути отримано з програмного коду і тестів.

Загальний код: будь-який член команди може в будь-який момент змінити будь-яку частину системи. Ця практика називалася в колишній версії XP «*колективне володіння кодом*».

Єдина база програмного коду: існує тільки одна офіційна версія розроблюваної системи. Якщо вам знадобилося створити для чогось її гілку, залишайте її лише на кілька годин.

Щоденна поставка системи: щоночі треба збирати нову версію системи і вводити її в дію. Чим більший розрив між офіційною версією системи і тією, що знаходиться у вас в комп'ютері, тим більші ризики і тим дорожче це для проекту.

Графік, показаний на рис. 8.1. показує, як правила екстремального програмування працюють разом. Клієнти люблять бути партнерами в процесі розроблення програмного забезпечення і активно сприяти незалежно від рівня досвіду, а менеджери мають бути зосередженими на зв'язках і відношеннях..

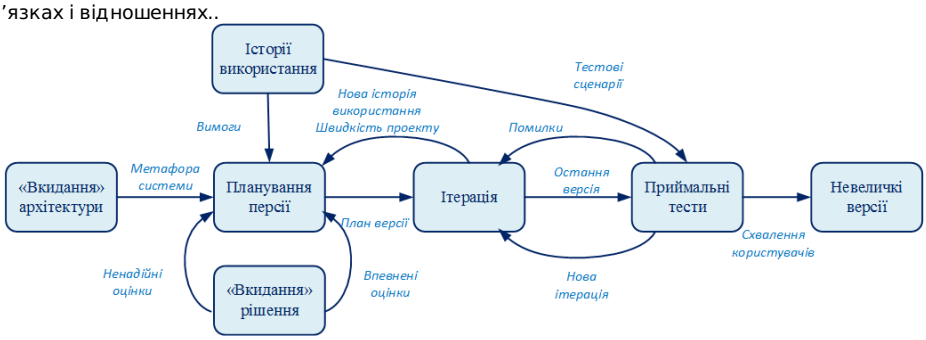


Рисунок 8.1 – Схема потоку робіт в XP

У результаті ми отримуємо метод, який потенційно володіє високою адаптацією до серйозно і часто мінливих вимог до проекту, але в той же час не позбавлений ряду принципових недоліків і дуже високого ступеня залежності від людського фактору.

Таким чином, результат застосування методу екстремального програмування може вийти або екстремально хорошим, або екстремально поганим.

ВИСНОВКИ

Екстремальне програмування є успішним, тому що це підкреслює задоволення клієнтів. Замість того, щоб поставляти все можливе на деякі дати в далекому майбутньому, цей процес пропонує необхідне програмне забезпечення, як і коли це потрібно. Екстремальне програмування дає змогу своїм розробникам впевнено реагувати на мінливі вимоги замовника, навіть наприкінці життєвого циклу.

Сформулюємо ключові напрямки, на які націлена XP:

- Робота з замовником. З'ясування того, що насправді потрібно замовнику. Як правило, це не те, що він собі уявляє. Своєчасне з'ясування подробиць і планування збереже багато сил, коштів, часу і нервів.
- Гнучка розробка. Горезвісний простий дизайн, часті випуски версій, регулярне планування – все це служить тому, щоб максимально швидко і безболісно реагувати на мінливі вимоги замовника і оперативно реалізовувати функціональність, найбільш критичну прямо зараз.
- Безперервна працездатність коду. Постійні інтеграції коду і велика кількість тестів – це все дає впевненість, що код працездатний і працює він правильно.
- Спрощення підтримки коду. Рефакторинг і Стандарти кодування полегшують подальші зміни в коді і полегшують розуміння коду усіма розробниками.
- Підвищення швидкості і якості розробки. Парне програмування, колективне володіння кодом, замовник в команді, 40-годинний робочий тиждень і метафора системи - ці практики роблять розробку більш швидкої і якісної.

Перевагами XP, якщо його вдається застосувати, є велика гнучкість, можливість швидко і акуратно вносити зміни в ПО у відповідь на зміни вимог і окремі побажання замовників, висока якість виходить в результаті коду і відсутності необхідності переконувати замовників у тому, що результат відповідає їх очікуванням.

Недоліками цього підходу є нездійсненність в такому стилі досить великих і складних проектів, неможливість планувати терміни і трудомісткість проекту на досить довгу перспективу і чітко передбачити результати тривалого проекту в термінах співвідношення якості результату і витрат часу і ресурсів. Також можна відзначити непристосованість XP для тих випадків, в яких можливі рішення не перебувають відразу на основі раніше отриманого досвіду, а вимагають проведення попередніх досліджень.

Для підготовки лекції були використані матеріали:
http://www.maxkir.com/sd/extreme_programming_2.html
http://www.info-system.ru/article/article_extrime_prog.html
19

Інформаційне забезпечення інформаційних систем

9.1 ПОНЯТТЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Для організації інформаційної взаємодії різноманітних інформаційних систем між собою, а також з різними групами користувачів дані потрібно відповідним чином однотипово описати в усіх системах на різних рівнях, тобто вирішити проблему їх інформаційної сумісності в найширшому розумінні. Цього досягають створенням інформаційного забезпечення, під яким розуміють сукупність (рис. 9.1.) форм документів, нормативної бази та реалізованих рішень щодо обсягів, розміщення і форм існування інформації, яка використовується в інформаційній системі при її функціонуванні.

Методичні та інструктивні матеріали – це сукупність державних стандартів, галузевих керівних методичних матеріалів і розроблених проектних рішень щодо створення й супроводження інформаційного забезпечення.

Системи класифікації і кодування – це перелік описів і систем супроводження класифікаторів техніко-економічної інформації на економічному об'єкті.

Рисунок 9.1 – Структура інформаційного забезпечення

Основні принципи створення інформаційного забезпечення: цілісність, вірогідність, контроль, захист від несанкціонованого доступу, єдність і гнучкість, стандартизація та уніфікація, адаптивність, мінімізація введення і виведення інформації (однократність введення інформації, принцип введення – виведення тільки змін).

Цілісність – здатність даних задовольняти принцип повного узгодження, точність, доступність і достовірне відображення реального стану об'єкта.

Існують два підходи до створення ІБ: аналіз сутностей; синтез атрибутів.

Аналіз сутностей є спадним підходом, або «згори – вниз», який поділяє процес створення на чотири стадії:

- моделювання уявлень користувачів;
- об'єднання уявлень;
- складання і аналіз моделі (схеми);
- реальне (фізичне) проектування.

Синтез атрибутів є зростаючим підходом, або «знизу – вгору», оскільки він починається із синтезу атрибутів найнижчого рівня, з яких формуються сутності та зв'язки верхнього рівня. Виділяють чотири стадії для цього підходу:

- класифікація атрибутів;
- композиція сутностей;
- формування зв'язків;
- графічне уявлення.

Кожний з цих підходів має свої переваги й недоліки і визначається виходячи із потреб проектування ІС. Для створення великих ІС, у яких є структура, найбільш прийнятний аналіз сутностей, для автономних невеликих ІС без структури – атрибутний (локальний).

Інформаційне забезпечення не можна успішно спроектувати без загального планування «згори – вниз» і детального проектування «знизу – вгору». Погодження двох підходів, в свою чергу, не можна досягти без відповідної методики, загальні аспекти якої ми розглядаємо.

Вимоги до інформаційного забезпечення (ГОСТ 24.104–85 «Автоматизированные системы управления. Общие требования») такі:

1. Інформаційне забезпечення має бути достатнім для виконання всіх функцій ІС, які автоматизуються.
2. Для кодування інформації, яка використовується тільки в цій ІС, мають бути застосовані класифікатори, які є у замовника ІС.
3. Для кодування в ІС вихідної інформації, яка використовується на вищому рівні, мають бути використані класифікатори цього рівня, крім спеціально обумовлених випадків.
4. Інформаційне забезпечення ІС має бути суміщене з інформаційним забезпеченням систем, які взаємодіють з нею, за змістом, системою кодування, методами адресації, форматами даних і формами подання інформації, яка отримується і видається інформаційною системою.
5. Форми документів, які створюються інформаційною системою, мають відповідати вимогам стандартів УСД чи нормативно-технічним документам замовника ІС.
6. Форми документів і відеокадрів, які вводяться, виводяться чи коригуються через термінали ІС, мають бути погоджені з відповідними технічними характеристиками терміналів.
7. Сукупність інформаційних масивів ІС має бути організована у вигляді бази даних на машинних носіях.
8. Форми подання вихідної інформації ІС мають бути погоджені із замовником (користувачем) системи.
9. Терміни і скорочення, які застосовуються у вихідних повідомленнях, мають бути загальноприйнятими в цій проблемній сфері й погоджені із замовником системи.
10. У ІС мають бути передбачені необхідні заходи щодо контролю і оновлення даних в інформаційних масивах ІС, оновлення масивів після відмови будь-яких технічних засобів ІС, а також контролю ідентичності однойменної інформації в базах даних.

Можуть створюватись також самостійні інформаційні засоби і вироби для конкретного користувача.

Інформаційний засіб – комплекс упорядкованої, відносно постійної інформації на носіях даних, які описують параметри та характеристики заданої проблемної сфери застосування, і відповідної документації, призначеної для поставки користувачеві.

Інформаційний виріб в ІС – виготовлений інформаційний засіб, який пройшов випробування встановленого вигляду та поставляється як продукція виробничо-технічного призначення для використання в ІС. Наприклад: словники, довідники підприємств і організацій, товарів, класифікатори, правові інформаційні системи і т.п.

Інформаційний продукт (продукція) – документована інформація, яка підготовлена і призначена для задоволення потреб користувачів.

Інформаційний ресурс – сукупність документів у інформаційних системах (бібліотеках, архівах, банках даних тощо).

9.2 ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОЇ БАЗИ

Ефективне функціонування інформаційної системи об'єкта можливе лише при відповідній організації інформаційної бази – сукупності впорядкованої інформації, яка використовується при функціонуванні ІС і поділяється на зонішньо- і внутрішньомашинну (машинну) бази (ГОСТ 34.003–90).

Зовнішньомашинна інформаційна база – частина інформаційної бази, яка являє собою сукупність повідомлень, сигналів і документів, призначених для безпосереднього сприйняття людиною без застосування засобів обчислювальної техніки.

Внутрішньомашинна інформаційна база – частина інформаційної бази, яка є сукупністю інформації, що використовується в ІС на носіях даних.

Така зовнішньомашинна ІБ має багато модифікацій від подання у вигляді повідомлень на паперовому носії, запитів на екрані дисплея та домовного спілкування з ЕОМ.

Внутрішньомашинна ІБ пройшла три етапи еволюції.

Перший етап характеризується роз'єднаним фондом даних:

- програми розв'язання кожної окремої задачі становили одне ціле з масивами, які оброблялися;
- використання якого-небудь масиву для іншої задачі забезпечувалось індивідуально пристосуванням до форм подання даних, структур елементів масивів і т ін.;
- опис даних не потрібний, оскільки структура була раніше відома;
- коригування масивів виконувалось індивідуальними засобами;
- задача розв'язувалася в пакетному режимі, користувач отримував результати винятково у вигляді машинограм і виробничих документів через групу підготовки і оформлення даних.

Дані розглядаємо на трьох рівнях, і є пряма залежність логічного рівня програми (ЛРП), фізичного (ФРЗ) та логічного (ЛРЗ) рівня збереження (ЛРП=ФРЗ=ЛРЗ).

Другий етап – централізований фонд даних.

1. Дані відокремлені від процедур їх обробки і організовані в бібліотеки масивів загального користування. Подання інформації, формати елементів даних і структура масивів уніфіковані і не залежать від конфігурації пам'яті та її організації.
2. Опис даних відокремлено як від програм, так і від самих даних, тому дані й програми їх обробки стають значною мірою незалежними. Це полегшує зміну структур даних і програм. Але реорганізація бібліотеки і її окремих груп компонентів потребує зміни програми обробки.

Залишаються залежні логічні рівні програми і збереження (ЛРП=ЛРЗ).

Третій етап – організація баз даних – характеризується:

1. Об'єднання не лише інформації, а й апаратно-програмних засобів її поповнення, коригування і видачі користувачеві.

3. Повне відокремленням функцій нагромадження, ведення і реорганізації даних від функцій їх обробки. Дані коригуються поза рівнем програм користувача за допомогою власного апарату бази даних.
4. Поява логічного буфера, системи управління базою даних, розв'язки між програмами користувача і базою даних.
5. Можливість оперативної реалізації довільних запитів у режимі безпосереднього зв'язку з ЕОМ.
6. Високий ступень централізації загальносистемних масивів, яка передбачає спільне використання загальних даних.
7. Різноманітність даних і зв'язаність в довільні логічні структури.
8. Наявність потужного програмного забезпечення і мовних засобів.

Усі рівні незалежні.

Нині існують другий і третій етапи. Основною задачею є визначення потрібної кількості баз даних і оптимального розподілу інформації між ними з урахуванням того, що економічний об'єкт – це динамічна система, яка перебуває в постійному розвитку. Використовуючи пріоритет виробничих функцій, необхідно побудувати таку базу даних. Так, навколо поняття «Модель виробу» формуються дві оболонки: внутрішня являє конструкторську документацію, зовнішня – технологічну і управлінську інформацію (рис. 9.2).

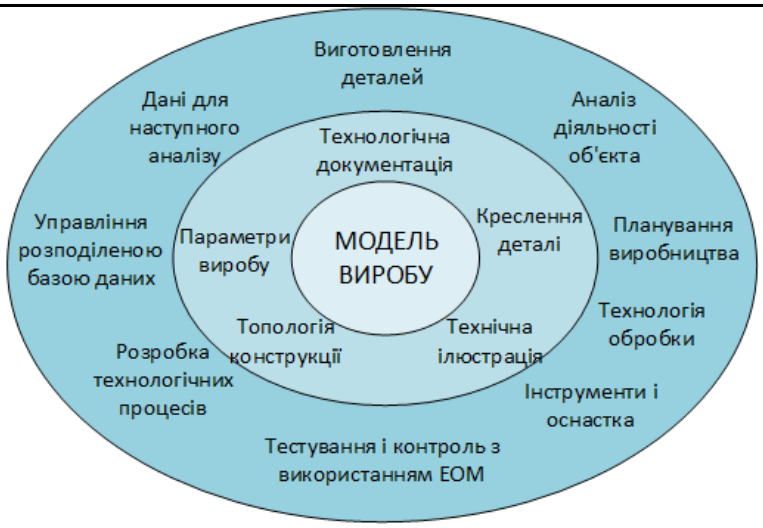


Рисунок 9.2 – Інформаційне забезпечення моделі виробу

Однак виникла така проблема: визначити, чи потрібна одна база даних, кілька локальних, взаємозв'язана розподілена база даних, локальні файли чи їх комбінації і т.п. При цьому враховується інформація, що використовується для реалізації багатьох функцій, особливо в оперативному режимі, активна інформація, тобто така, що використовується багаторазово.

Описуючи організацію інформаційної бази, потрібно дати опис логічної і фізичної структур бази даних.

Документ складається з двох частин:

1) опис внутрішньомашинної інформаційної бази;
2) опис зовнішньомашинної інформаційної бази.

Кожна частина складається з таких розділів:

- логічна структура;
- фізична структура (для зовнішньомашинної інформаційної бази);
- організація ведення інформаційної бази.

У розділі «Логічна структура» наводять опис складу даних, їх формати і взаємозв'язки між даними.

У розділі «Фізична структура» наводять опис вибраного варіанта розміщення даних на конкретних машинних носіях даних.

При описі структури внутрішньомашинної інформаційної бази наводять перелік баз даних і масивів та логічні зв'язки між ними. Для масиву інформації вказують логічну структуру масиву чи дають посилання на документ «Опис масиву інформації».

Описуючи структуру зовнішньомашинної інформаційної бази, наводять перелік документів та інших інформаційних повідомлень, використання яких передбачено в системі, із зазначенням автоматизованих функцій, при реалізації яких формується чи використовується цей документ.

Якщо цю інформацію наведено у документах «Перелік вхідних сигналів і даних» і «Перелік вихідних сигналів», можна посилатися на ці документи.

У розділі «Організація ведення інформаційної бази», описуючи внутрішньомашинну базу, наводять послідовність процедур при створенні і обслуговуванні бази із зазначенням в разі потреби регламенту виконання процедур і засобів захисту бази від руйнування і несанкціонованого доступу, а також зв'язків між масивами баз даних і масивами вхідної інформації.

Описуючи зовнішньомашинну інформаційну базу потрібно навести послідовність процедур по маршруту руху груп документів до передачі їх на обробку, а також описати маршрут руху вихідних документів

9.3 ВИДИ ІНФОРМАЦІЙНИХ МАСИВІВ

При організації раціонального варіанта внутрішньомашинної інформаційної бази даних, яка найбільш повно відбиває специфіку об'єкта управління, перед розробниками постають вимоги до організації масивів, які можуть бути суперечливими. До них належать:

- повнота подання даних;
- мінімальний склад даних;
- мінімізація часу вибірки даних;
- незалежність структури масивів від програмних засобів їх організації;
- динамічність структури інформаційної бази.

Найбільш суперечливою з них є вимога повноти подання даних, мінімізація складу даних і мінімізація часу вибірки даних. Оптимальним є повне взаємне врахування всіх вимог, що випливають з процесів, які автоматизуються.

Останнім часом склалися такі основні підходи до побудови внутрішньомашинної інформаційної бази:

- проектування масиву як відображення змісту окремого документа;
- проектування масивів для окремих процесів управління;
- проектування масивів для комплексів процесів управління, які реалізуються;
- проектування бази даних;
- проектування кількох баз даних.

Кожний з цих підходів має свої переваги і недоліки, а вибір залежить від обчислювальної техніки, яка використовується, програмних засобів і специфіки процесів, що автоматизуються.

Проектування масивів передбачає визначення їх складу, змісту, структури і вибір раціонального способу їх подання в пам'яті обчислювальної системи.

Поняття складу і змісту масивів передбачає визначення оптимальної кількості масивів і

переліку атрибутів (полів), які у них містяться.

Під структурою масиву розуміємо формат записів у масиві, розмір полів і їх розміщення в машинному записі, ключові атрибути і впорядкування масиву за ними.

Вибираючи раціональний спосіб подання масиву в пам'яті визначають такий спосіб зберігання даних, за якого забезпечувалися б мінімальний обсяг пам'яті для розміщення масиву, висока швидкість пошуку даних, а також можливість збільшення і оновлення масиву. Кожний масив характеризується обсягом, способом організації, стабільністю і ступенем активності.

З точки зору використання масивів на різних етапах технологічного процесу обробки даних виділяють такі типи масивів: вхідні (первинні), основні (базові), робочі (проміжні) й вихідні (результатні).

Вхідні масиви – це проміжна ланка між первинними інформаційними повідомленнями і основними масивами. Зміст і розміщення даних у вхідному масиві аналогічні змісту й розміщенню їх у первинному інформаційному повідомленні.

Основні масиви створюються на основі вхідних, постійно зберігаються і містять основні дані про об'єкти управління і процеси виробництва. Кожний основний масив містить усю сукупність інформації, яка всебічно характеризує однорідні об'єкти і потрібна для реалізації функцій управління. За змістом ці масиви ми можемо класифікувати на такі групи: нормативні, розціночні, планово-договірні, регламентуючі, довідково-табличні й постійно-облікові.

Необхідність створення таких масивів зумовлена необхідністю забезпечення принципу одноразового формування масивів, внесення змін і усунення дублювання. Це в свою чергу призводить до різкого збільшення його розміру і ускладнення використання в процесі реалізації тих чи інших процесів, оскільки часто потрібна лише частина інформації основного масиву, а це вимагає створення робочих масивів.

Робочі масиви призначені для роботи програм, які реалізують розв'язання конкретних задач процесів управління і містять обмежене коло атрибутів одного чи кількох основних масивів. Робочі масиви організуються в момент розв'язання задачі й лише на час її розв'язання, після чого їх анулюють.

Вихідні масиви формуються в процесі розв'язання задачі й використовуються для модифікації основних масивів і виведення вихідних (результатних) інформаційних повідомлень.

Основні масиви можуть мати вигляд локальних масивів чи організовані в базу даних (БД) під керуванням системи управління базою даних (СУБД).

Взаємозв'язок користувача з базою даних зображено на рис. 9.3.

База даних – іменована сукупність даних, що відображає стан об`єктів та їх відношення у визначеній проблемній сфері (закон України "Про Національну програму інформатизації" (74/98-ВР від 04.02.98))

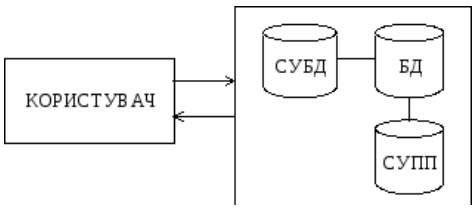


Рисунок 9.3 - Взаємозв'язок користувача з базою даних

Система управління базами даних – це сукупність програм і мовних засобів, які призначені для управління даними в базі даних і забезпечують взаємодію її з прикладними програмами.

Масив даних – це конструкція даних, компоненти якої ідентичні за своїми характеристиками і є значеннями функції від фіксованої кількості цілочислових аргументів.

Файл – це ідентифікована сукупність примірників повністю описаного в конкретній програмі типу даних, розміщених зовні програми в зовнішній пам'яті та доступних програмі, за допомогою спеціальних операцій.

9.4 МЕТОДИКА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Методика проектування інформаційного забезпечення складається з трьох етапів.

На *першому етапі* «*Розробка рішень по інформаційній базі*»: визначається склад і обсяг нормативно-довідкової інформації; розробляються пропозиції щодо вдосконалення діючого документообігу; структура бази даних; система збирання і передавання інформації, а також рішення з організації і ведення бази даних; визначається склад і характеристики вихідної і вхідної інформації (сигналів, документів, даних).

На *другому етапі* «*Вибір номенклатури і прив'язка системи класифікації і кодування інформації*»: визначається перелік типів інформаційних об'єктів, які підлягають ідентифікації в ІС, перелік необхідних класифікаторів; вибираються й розроблюються класифікатори інформаційних об'єктів і системи кодування; визначається система внесення змін і доповнень у класифікатори; розробляються принципи й алгоритми автоматизованого ведення класифікаторів.

На *третьому етапі* «*Розробка рішень щодо забезпечення обміну інформацією в системі*» розробляється схема інформаційного забезпечення

ВИКОРИСТАНА ЛІТЕРАТУРА

Для підготовки матеріалу були використані такі джерела інформації:

1. Береза А.М. Основи створення інформаційних систем: Навч. посібник. 2 видання,перероблене і доповнене – К.: КНЕУ, 2001

- Використана література

10.1 ОСНОВНІ ПОНЯТТЯ КЛАСИФІКАЦІЇ ІНФОРМАЦІЇ

Методи організації і пошуку керуючої інформації в умовах її автоматизованої обробки потребують попередньої класифікації і кодування.

Класифікація – обов'язковий етап попередньої підготовки даних до автоматизованої обробки, а також передумова раціональної організації інформаційної бази (ІБ) і моделювання інформаційних процесів. Її можна схарактеризувати як складову інформаційного забезпечення будь-якої інформаційної системи, яка належить до мовних засобів управління.

Тому класифікація – поділ множини об'єктів на підмножини за їх подібністю або відмінністю згідно з прийнятими методами класифікації – і є основою для кодування інформації і наступного її пошуку за допомогою обчислювальної техніки.

Під класифікацією інформації розуміємо не лише інформацію, яка є у масивах і повідомленнях, а й класифікацію безпосередньо інформаційних повідомлень (документів) і масивів.

Система класифікації є сукупність методів і правил класифікації та її результат.

Об'єкт класифікації – елемент класифікаційної множини (предмети, поняття, властивості тощо).

Ознака (критерій) класифікації – властивість чи характеристика об'єкта, за яким здійснюється класифікація. Кількісні та якісні вирази ознаки класифікації є її значенням.

Класифікаційне групування – підмножина об'єктів, які отримані в результаті класифікації.

Залежно від того, як розглядається дана множина об'єктів - послідовно чи одночасно за всіма ознакам основи поділу, використовують ієрархічний чи фасетний метод класифікації.

10.1.1. Ієрархічний метод класифікації.

Ієрархічний метод класифікації – послідовний поділ множини об'єктів на підлеглі класифікаційні групування.

Множину, яка класифікується, поділяють на підпорядковані підмножини спочатку за деякою ознакою (основою поділу) на великі групування, потім кожну з них - на ряд наступних угрупувань, які в свою чергу поділяють на дрібніші, поступово конкретизуючи об'єкт класифікації. Між цими угрупуваннями встановлюються відношення підпорядкованості (ієрархії) (рис. 10.1).

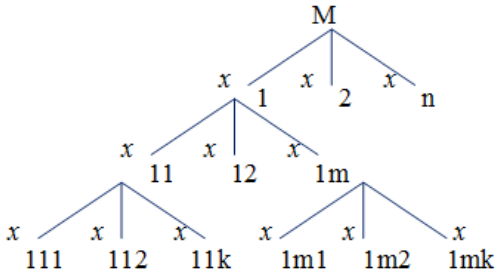


Рисунок 10.1 – Схема побудови коду за ієрархічним методом класифікації

Ієрархічна класифікація характеризується кількістю ступенів класифікації, глибиною, обсягом і гнучкістю.

Сукупність класифікаційних угрупувань є ступенем класифікації.

Кількість ступенів класифікації визначає глибину класифікації, яку встановлюють залежно від ступеня конкретизації групування і кількості ознак, необхідних для розв'язання конкретних задач.

Від глибини класифікації й кількості угрупувань, які створюються на кожному ступені класифікації, залежить обсяг класифікації.

Як правило, найбільша кількість угрупувань, на яку може поділятися дане групування, що встановлюється постійним для всієї класифікації чи для даного ступеня, звичайно є кратною десяти.

Переваги: логічність побудови, чіткість виділення ознак, великий інформаційний обсяг, традиційність і звичність використання, добра пристосованість для ручної обробки інформації, можливість створення мнемонічних кодів, які несуть смислове навантаження.

Недоліки: жорстка структура, зумовлена фіксованістю ознак і заздалегідь встановленим порядком їх проходження, які не допускають включення за відсутності резервного обсягу нових об'єктів, класифікаційних угрупувань і ознак; неможливість групувати за будь-якою, наперед не заданою ознакою; для стабільності класифікаторів потрібні великі резервні обсяги.

10.1.2. Фасетний метод класифікації.

Фасетний метод класифікації – паралельний поділ множини об'єктів на незалежні класифікаційні групування.

При цьому множина об'єктів, що характеризується деяким набором однакових для всіх об'єктів ознак (фасет), значення яких відповідають конкретним виразам зазначених ознак, може поділятися багаторазово і незалежно. У класифікаторах фасети найчастіше розміщуються простим переліком і мають свій код (рис. 10.2).

Класифікаційні групування створюються з об'єктів, які мають конкретні комбінації ознак, взяті з відповідних фасет. Послідовність розміщення фасет при створенні класифікаційного групування задається фасетною формулою

$$G=\langle F_1, F_2, \dots, F_r, F_R \rangle .$$

У кожному окремому випадку фасетна формула визначається залежно від характеру розв'язуваних задач і алгоритму обробки даних. Можуть створюватись одночасно різні незалежні підмножини класифікаційних угрупувань:

$$X_1=\langle F_1, F_2, \dots, F_r, F_R \rangle,$$
$$X_2=\langle F_1, F_2, F_R \rangle,$$
$$X_3=\langle F_1, F_2 \rangle,$$
$$X_{n-1}=\langle F_1, F_R \rangle.$$

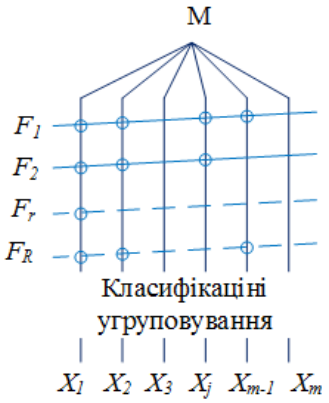


Рисунок 10.2 – Схема побудови коду за фасетним методом класифікації

Обсяг залежить від кількості фасет і кількості конкретних значень ознак у фасеті. Фасети у створюваному класифікаторі мають строго фіксоване місце. Їх ідентифікують за кодовим позначенням фасета, найчастіше це його порядковий номер.

Переваги: гнучкість структури, яка може пристосовуватися до змін у задачах; можна включати нові фасети чи видаляти старі.

Недоліки: недостатньо повне використання обсягу через відсутність практично багатьох із можливих комбінацій фасет; нетрадиційність і незвичність при використанні для ручної обробки даних.

10.1.3. Вимоги до методу класифікації.

Вибраний метод класифікації має задовольняти такі вимоги:

1. Мати достатній обсяг і необхідну повноту, які б гарантували охоплення всіх об’єктів класифікації в заданих межах.
2. Не перетинати груп об’єктів, які виділяються.
3. Мати достатню та обґрунтовану глибину.
4. Мати гнучкість і надмірність для можливого збільшення множини об’єктів, які класифікуються.
5. Забезпечувати розв’язання всього комплексу задач.
6. Забезпечувати сполучення з іншими класифікаціями однорідних об’єктів.
7. Бути погодженим з алгоритмами і забезпечувати найбільшу ефективність обробки.
8. Забезпечувати простоту і автоматизацію процесу ведення класифікатора.
9. Лаконічність, чіткість і ясність класифікаційних ознак.

10.2 КОДУВАННЯ ІНФОРМАЦІЇ

У процесі кодування об’єктів класифікації їх групуванням і ознакам за певними правилам присвоюють цифрові, літерні чи літеро-цифрові коди.

Кодування – утворення і присвоєння коду класифікаційному групуванню чи об’єкту класифікації.

Система кодування – це сукупність методів і правил кодування класифікаційних угрупувань і об’єктів класифікації заданої множини.

Код – це знак чи сукупність знаків, прийнятих для позначення класифікаційного групування чи об’єкта класифікації.

Код і його структура характеризуються алфавітом, основою і довжиною.

Структура коду – це умовне позначення складу і послідовності розміщення знаків у коді.

Розряд коду – позиція знаку в коді.

Алфавіт коду – система знаків, яка прийнята для утворення коду.

Основа коду – кількість знаків у алфавіті коду.

Довжина коду – кількість знаків у коді без урахування пропусків.

Є чотири методи кодування: порядковий, серійно-порядковий, послідовний і паралельний.

Усі ці методи розглянемо на прикладі кодування студентів академічної групи з визначенням ознаки статі. Дані розмістимо в табл. 10.1.

Порядковий метод кодування – створення коду із чисел натурального ряду і його привласнення. Найбільш простий і повний, однозначний.

На основі максимальної кількості об’єктів, які класифікуються, визначається кількість розрядів для ознаки і всього коду.

Так, у групі 25 студентів – потрібна довжина коду при десятковій основі у два розряди. Присвоїмо коди студентам у таблиці.

Серійно-порядковий метод кодування – створення коду із чисел натурального ряду, із закріпленням окремих серій чи діапазонів цих чисел за об’єктами класифікації з однаковими ознаками і його привласнення. Використовується для двознакових номенклатур.

При визначенні кількості розрядів для коду беруть до уваги максимальну кількість об’єктів для найбільшої серії чи діапазону і добавляють резервні позиції для кодування нових об’єктів. Їх кількість визначають на основі обстеження проблемної сфери чи беруть 25% найбільшої кількості об’єктів.

У групі 17 дівчат і 8 хлопців. Для кодування найбільшої серії (дівчата) потрібен код із двох розрядів.

1. - хлопців 01 - 08, 3 розряди резервні. Тому 01 - 11.
2. - дівчат 12 - 29, 4 розряди резервні. Тому 12 - 33.
3. Присвоїмо коди студентам у таблиці

Ці два коди повністю ідентифікують об’єкт, але не віддзеркалюють ознакову інформацію про нього в коді і здебільшого використовуються для передавання інформації на відстані. Їх особливість - незалежність від методів класифікації, які використовуються, і суті розв’язуваних задач, складність при автоматизованій обробці (групування з подібними ознаками, підсумовування за групою об’єктів).

Таблиця 10.1 – Приклад кодування інформації. Кодування академічної групи з визначенням ознаки статі

Прізвища студентів	Порядковий	Серійно-порядковий	Послідовний	Паралельний
Абрамов	01	01	101	101
Сидорова	02	12	201	202
Рогачова	03	13	202	203
Борисов	04	02	102	104
…				
Шутов	24	08	108	124
Юрова	25	29	217	225
Волошин	26	09	109	126

Послідовний метод кодування – це створення коду класифікаційного групування і (чи) об'єкта класифікації з використанням кодів послідовно розміщених підпорядкованих угрупувань, які отримані при ієрархічному методі класифікації, і його привласнення.

Переваги послідовного методу: простота побудови коду, велика місткість при великій інформативності, можливість отримання результатів по вищих (старших) розрядах.

Недоліки: велика кількість знаків у коді і складність побудови задач.

Паралельний метод кодування – це створення коду класифікаційного групування і (чи) об'єкта класифікації з використанням кодів незалежних угрупувань, які отримані при фасетному методі класифікації, і його привласнення.

Переваги паралельного методу: добра пристосованість для автоматизованої обробки і розв'язання техніко-економічних задач, характер яких постійно змінюється, фасетна побудова уможливлює стандартизацію.

Недоліки: обмежені можливості ідентифікації об'єктів, велика надмірність, неповне використання обсягу створеної класифікації.

Ці методи дають істотну ознакову інформацію про об'єкт, але мають обмежені можливості ідентифікувати їх.

Як методи класифікації, так і методи кодування самостійно практично не застосовуються. Аби скористатися перевагами різних методів, на практиці використовують різні комбінації методів класифікації та кодування.

Можливі комбінації визначаються їх взаємозв'язком, який дає певні конкретні структури, що використовуються у відповідних класифікаторах. Вибір тієї чи іншої комбінації залежить від призначення класифікатора і конкретних задач, у яких він буде використовуватись.

Для формування документації з інформаційного забезпечення складається документ «Побудова системи класифікації і кодування» і «Опис систем класифікації і кодування», в якому по кожному об'єкту, який класифікується, має бути наведено опис методу кодування, структуру і довжину коду, вказівки про систему класифікації та інші відомості на вибір розробника.

Вимоги до кодів:

- забезпечення розв'язання всіх задач системи при мінімумі їх довжини;
- єдність кодів на всіх рівнях управління;
- структура коду має забезпечувати групування інформації у необхідних розрізах;
- зміст номенклатур повинен відповідати вимогам державних стандартів чи керівних методичних матеріалів;
- забезпечення інформаційного сполучення взаємопозв'язаних систем;
- автоматичний контроль помилок.

Кодування інформації виконується такими способами:

- ручним проставленням (присвоєнням) у повідомленнях того чи іншого коду поряд з назвою об'єктів номенклатур;
- друкарським – ряд номенклатур кодується в процесі виготовлення бланків носіїв інформації (код складу, код операції руху матеріалів, код документа і т.п.);
- автоматизованим – виведенням коду на екран з масиву;
- на спеціальному обладнанні, яке дає змогу автоматично кодувати інформацію.

Потрібно створювати спеціальні програмні засоби, які автоматизують процес їх побудови і використання.

10.3 КЛАСИФІКАТОРИ ТЕХНІКО-КЕРУЮЧОЇ ІНФОРМАЦІЇ

Для інформаційної сумісності різних систем потрібно використовувати однакові класифікатори. Класифікатор – це офіційний документ, який являє собою систематизований перелік найменувань і кодів класифікаційних угрупувань і (чи) об'єктів класифікації.

Позиція - найменування і код класифікаційного групування і (чи) об'єкта класифікації.

Місткість класифікатора – найбільша кількість позицій, які може мати класифікатор.

Резервна місткість класифікатора – кількість вільних позицій у класифікаторі.

Класифікатори можуть бути: державні (затверджені Держстандартом для використання в ІС різних міністерств і відомств); галузеві (введені в установленому порядку для використання в ІС даної галузі); підприємств (уведений в установленому порядку для використання в ІС цього підприємства).

Для цього створено Єдину систему класифікації і кодування техніко-керуючої інформації (ЄСКК).

Єдина система класифікації і кодування техніко-керуючої інформації ((СКІК є частиною інформаційного забезпечення системи і становить сукупність взаємопов'язаних державних класифікаторів техніко-керуючої інформації, системи ведення і керівних нормативних документів для їх розробки, впровадження, ведення, вдосконалення й контролю за впровадженням.

Створенням ЄСКК керує Держстандарт України. ЄСКК містить понад 20 класифікаторів, які можемо поділити на 4 групи:

- класифікатори інформації про трудові та природні ресурси, професії робітників і посади службовців за категоріями, спеціальностями, освітою і т. п.;
- класифікатори інформації про продукти праці, виробничу діяльність та послуги, промислову та іншу продукцію, роботи й послуги в різних галузях;

- класифікатори інформації про структуру народного господарства і адміністративний поділ, підприємства, організації, галузі народного господарства і т. п.;
- класифікатори управлінської інформації і документації, позначення одиниць вимірювання, техніко-економічних показників, стандартів тощо.

Наприклад:
код 021124 02 – нафтопродукти

1. – нафтопродукти світлі 0211 - бензин
2. – бензин автомобільний
3. – бензин автомобільний марки А-72

Державний класифікатор продукції та послуг має структуру, зображена на рис. 10 3.:

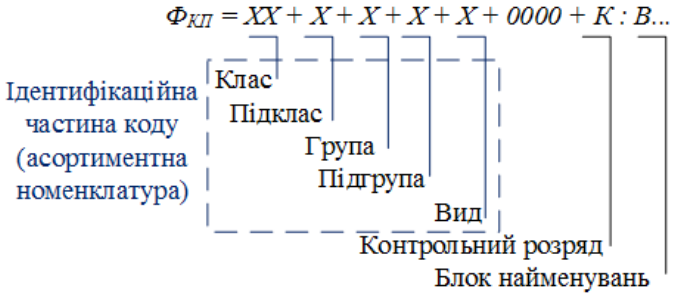


Рисунок 10.3 – Структура коду державного класифікатора продукції та послуг

Державний класифікатор підприємств і організацій має таку структуру (рис. 10.4):

У ньому використано окрему ідентифікацію і класифікацію. Структурно він складається з трьох блоків: ідентифікації, назв і фасет класифікаційних ознак.

Блок ідентифікації – це перелік реєстраційних номерів підприємств і організацій, побудований за двоступеневим ієрархічним методом класифікації, з використанням серійно-порядкового методу кодування. Старша ознака – галузь народного господарства, в межах галузі розміщені порядкові номери об'єктів. Один розряд займає контрольна частина коду.

Блок назв складається з повної офіційної назви та вказівки його розміщення, які описані природною мовою й мають невизначену довжину.

Блок фасет класифікаційних ознак відбиває багатоступеневу класифікацію підприємств і організацій, побудовану із застосуванням фасетного методу класифікації і паралельного методу кодування цифровими десятковими знаками. Він містить три фасети:

$Ф_1$ – підпорядкованості, довжиною 8 розрядів, побудований за чотириступеневим ієрархічним методом класифікації та послідовним і серійно-порядковим методами кодування;

$Ф_2$ – адміністративно-територіальної підпорядкованості, має довжину 4 розряди і побудований за двоступеневим ієрархічним методом класифікації і порядковим методом кодування;

$Ф_3$ – галузевої підпорядкованості, побудований з використанням кодів державного класифікатора галузей народного господарства, довжиною 5 розрядів.

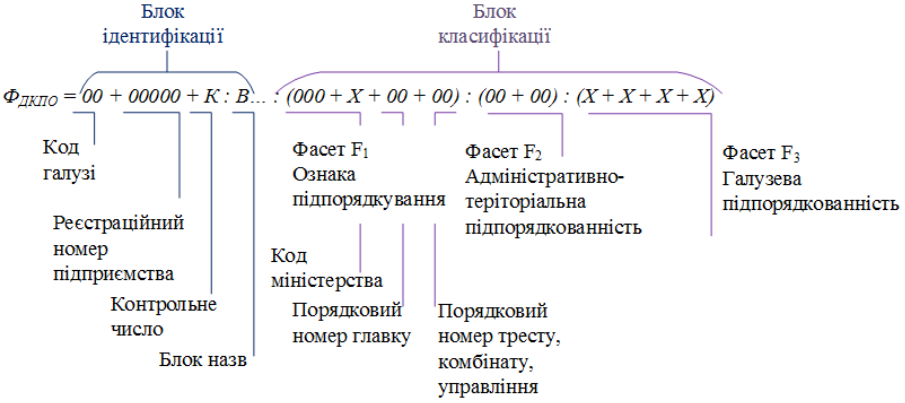


Рисунок 10.4 – Структура коду державного класифікатора підприємств і організацій

10.4 МЕТОДИКА СТВОРЕННЯ КЛАСИФІКАТОРІВ

Процес створення класифікаторів керуючої інформації можемо розглядати як сукупність організаційних, технічних і економічних заходів. Методика створення складається з чотирьох етапів:

1-й етап – на основі матеріалів обстеження об'єкта і технічного завдання розроблюють перелік необхідних класифікаторів.

2-й етап – на основі переліку необхідних класифікаторів, методичних матеріалів, вимог системи користувача і загальних параметрів створення інформаційної бази будують систему класифікації і кодування. При цьому:

- визначають найбільш суттєві ознаки, за якими систематизуються інформаційні сукупності в конкретному класифікаторі.
- визначають значність об'єктів, які будуть кодуватися.
- обирають метод класифікації і систематизують інформацію в класифікаторі.
- обирають метод кодування, розроблюють структуру коду і надають коди кожній позиції номенклатури.

Загальний перелік номенклатур оформлюють у вигляді класифікаторів, у яких розміщують

коротку інструкцію про порядок застосування кодів.

Розроблені класифікатори погоджують і затверджують у замовника і призначають підрозділи і осіб, відповідальних за ведення класифікатора.

Закінчують розробкою опису класифікатора і технічного завдання на класифікатор.

1-й етап – на основі опису класифікатора, технічного завдання на його створення, опису програмної системи чи документації на ППП розроблюють чи прив'язують ППП. На виході етапу маємо технологічну документацію на створення і ведення класифікатора, ППП.

2-й етап – на основі технологічної документації, ППП і матеріалів обстеження об'єкта створюють класифікатор у пам'яті ЕОМ, який роздруковується і передається відповідним підрозділам для користування.

Також розроблюється технологія ведення класифікаторів, де потрібно вирішити питання внесення нових записів, корекція старих, при необхідності видалення окремих записів, а при необхідності збереження старих записів (ведення бази даних з історією)

ВИКОРИСТАНА ЛІТЕРАТУРА

Для підготовки матеріалу були використані такі джерела інформації:

1. Недашківський О.М.. Планування та проектування інформаційних систем. – Київ, 2014. – 215 с.

11.1 ПОНЯТТЯ СИСТЕМИ ДОКУМЕНТАЦІЇ

В умовах інформаційної революції виникла проблема забезпечення сумісності інформаційних систем на різних рівнях, а тому важливе значення мають розробка і впровадження в різних галузях уніфікованих систем інформаційних повідомлень, які складаються з документів на папері, відеокадрів тощо.

Система документації – комплекс взаємопов'язаних документів, які необхідні для управління об'єктом управління.

Документ – матеріальний об'єкт, який містить у зафіксованому вигляді інформацію, оформлену в установленому порядку.

Під уніфікованою системою інформації розуміємо систему документів, яка є раціонально організованим комплектом взаємопов'язаних документів, що відповідають єдиним правилам і вимогам й містять інформацію, необхідну для оптимізації управління на основі використання ЕОМ і ОТ.

Документацію, яка використовується на всіх рівнях управління народного господарства, уніфіковують створенням:

- уніфікованих систем документації державного призначення (державні стандарти і міжгалузеві уніфіковані форми документів державного призначення, які відповідають вимогам цих стандартів);
- уніфікованих форм документів інших рівнів, які розроблюються на базі уніфікованих систем документації державного призначення (галузеві, відомчі й форми документів підприємств і організацій).

Уніфікацію документів потрібно виконувати на етапі проектування, вдосконалення і розвитку ІС з урахуванням усіх задач, які вони розв'язують.

У державі діють понад 16 уніфікованих систем, які містять 6 тис. форм документів державного призначення. Але в ІС застосовується щонайбільше 10 – 15%, оскільки більшість форм не пристосовані для обробки на ЕОМ.

11.2 КЛАСИФІКАЦІЯ ФОРМ І МЕТОДІВ ВИВЕДЕННЯ ІНФОРМАЦІЇ

Проектування форм і засобів виведення та відображення інформації у системах обробки даних і ІС є важливим питанням при створенні систем.

Вихідна інформація ІС – це інформація, яка видається на об'єкт управління, персоналу чи в інші системи управління, у вигляді документів, відображень, даних і сигналів і отримується в результаті виконання функцій інформаційної системи.

Вихідну керуючу інформацію необхідно згрупувати в необхідних напрямках за певними ознаками і після попередньої обробки звести до певних форм чи графіків.

Форми носіїв вихідної інформації, оброблюваної за допомогою обчислювальної техніки, залежать від експлуатаційних можливостей технічних засобів, які застосовуються, варіанта і повноти обробки цієї інформації, її призначення і методів використання.

Класифікація форм носіїв вихідної інформації за:

1. Характером – функціональні чи відображення окремих операцій.
2. Формою подання інформації – цифрова, алфавітно-цифрова, графічна.
3. Призначенням – основні, допоміжні.
4. Місцем використання – внутрішні, зовнішні.
5. Офіційністю – затверджені, незатверджені;
6. Періодичністю – поточні, квартальні, річні;
7. Термінами – оперативні, звичайні, нетермінові;
8. Характером змісту – змісту повідомлень, підсумкові;

9. Характером друку – широкий, вузький;
10. Кількістю примірників – один, два і т.д.

До проектування форм вихідної інформації висувають такі вимоги:

- наявність усіх необхідних показників, які встановлені відповідно до цільових функцій управління;
- максимальна закінченість, виключення будь-якого додаткового опрацювання вручну;
- точність підрахунків і зрозумілість друку;
- зручність і доступність використання широким колом осіб, які не ознайомлені з обчислювальною технікою;
- текстова розшифровка деяких якісних атрибутів;
- форма вихідного повідомлення і його зміст не повинні ускладнювати технологію його складання;
- форми документів і відеокадрів, які вводяться, виводяться чи коригуються через термінали ІС, мають бути погоджені з відповідними технічними характеристиками терміналів;
- форми подання вихідної інформації ІС мають бути погоджені із замовником (користувачем) системи;
- терміни і скорочення, які вживаються у вихідних повідомленнях ІС, мають бути загальноприйнятими в даній проблемній сфері й погоджені із замовником системи;
- об'єднання двох чи більше простих форм (з однаковою періодичністю складання) в одну форму з урахуванням можливостей ЕОТ;
- чітка назва вихідного повідомлення.

У процесі проектування вихідних повідомлень необхідно враховувати:

- цілі, для яких вони використовуються;
- сфери і особливості їх використання;
- періодичність отримання;
- можливості засобів виведення інформації;
- характер проблемної сфери;
- умови роботи з повідомленнями;
- контроль вірогідності складання;
- порядок оформлення і передавання користувачеві.

11.3 МЕТОДИКА ПРОЕКТУВАННЯ ФОРМ ВИХІДНОЇ ІНФОРМАЦІЇ

Методика створення форм вихідної інформації складається з 4 етапів.

1-й етап. Визначення загального складу зведених показників, які виводяться за допомогою обчислювальної техніки.

На цьому етапі у результаті аналізу встановлюється вся сукупність вихідних показників, їх важливі характеристики і вимоги користувача.

За цими матеріалами визначаються необхідні для об'єкта зведені показники і їх характеристики, а також можливість отримання їх за допомогою ЕОМ. Встановлення складу зведених показників дуже важливе у визначенні масивів бази даних. Тому одним із суттєвих принципів створення ІС є первісність вихідного повідомлення.

Складність робіт на цьому етапі полягає в необхідності врахування як вимог усіх рівнів управління у взаємозв'язку, так і для об'єкта в цілому.

2-й етап. Встановлення змісту інформації, яка входить до окремих вихідних повідомлень.

Встановлення однорідних показників, які потрібно отримати в однакові терміни використання і в одному підрозділі.

Якщо вони застосовуються в кількох підрозділах, то визначається кількість примірників.

Об'єднання кількох простих в один з використанням можливостей обчислювальної техніки.

Визначення атрибутів кожного вихідного повідомлення.

Розробка розділу 2 документа «Опис постановки задачі».

Необхідно передбачити найзручніший спосіб контролю за правильним їх складанням: спосіб балансування окремих граф, виведення додаткових контрольних підсумків, інструкції з контролю.

3-й етап. Розробка ескізу форми кожного вихідного повідомлення.

Може розроблятися така таблиця, де визначаються підпорядкованість ознак і кількість отриманих підсумків.

Для кожної форми виведення інформації визначається зміст трьох зон:

Зона 1 – заголовок чи титульний аркуш вихідного повідомлення, який виводиться одноразово чи частково повторюється на кожному аркуші (назва вихідного повідомлення, структурного підрозділу, період (дата), за який створюється даний документ, дата створення, місце підпису відповідальної за випуск документа особи);

Зона 2 – назва граф і їх нумерація, які виводяться на кожному аркуші (заголовки граф);

Зона 3 – основна предметна (інформаційна) частина, яка складає тіло вихідного повідомлення. До цієї зони включають рядки трьох видів: детальні, підсумкові чи їх суміш.

Детальні рядки містять дані, які є у записах масивів. Їх розміщують у такому порядку (рис. 11.1): спеціальні, довідкові, групові та довідкові змінні, показники з масивів.

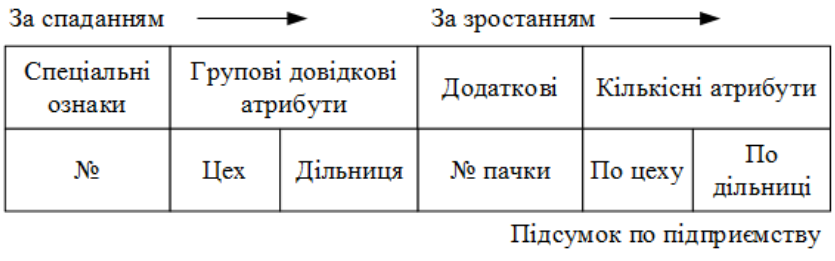


Рисунок 11.1 – Розміщення атрибутів у вихідному повідомленні

Підсумкові рядки містять кількісні й вартісні підсумки за певними ознаками, результати розрахунків. Вони розміщуються в основному за зростанням групових ознак. Їх розміщення залежить від кількості контрольованих ознак, які передбачені умовами виведення проміжних результатів (наприклад, по кожному аркушу) та інших факторів, що визначаються замовником і

користувачем. Кількісно-вартісні атрибути і показники розміщуються в такому порядку: нормативно-розцінкові, планові, фактичні й розрахункові. До підсумкових рядків різних ступенів необхідно додатково включати текстові коментарі (наприклад, «усього за табельним номером...», «усього за складом»)

Попередньо розраховують розміри граф, рядків, окремих рядків і всієї площі форми. Площу форми визначають з урахуванням розмірів окремих елементів формулярів, максимальної значності чисел для кожної графи, а також ширини інтервалів і літер друкуючого пристрою.

Після цього вибирають стандартний формат, додатково коригують окремі розміри, виготовляють зразок, викреслюючи його в натуральну величину.

Далі визначають, для яких випадків призначена форма, як нею користуватися, які документи замінює, порядок зберігання, які посадові особи або хто відповідає за правильність складання, і підписують.

4-й етап. Погодження із зацікавленими службами, внесення змін і затвердження у відповідальних осіб.

Зміст і форму кожного вихідного повідомлення визначають, ураховуючи цілі, для яких воно призначене, сферу та особливості його використання, періодичність отримання і т. ін. При цьому потрібно враховувати особливості технічних засобів, програмного забезпечення, можливість попереднього запису вихідної інформації на магнітні носії, використання діалогового режиму тощо.

11.4 ЗАГАЛЬНІ ВИМОГИ ДО ПРОЕКТУВАННЯ ФОРМ ПЕРВИННИХ ДОКУМЕНТІВ

Проектуванню форм вхідної інформації приділяється особлива увага. На цьому етапі можна скоротити обсяги даних, трудові витрати на збирання, реєстрацію, передавання і перенесення на машинні носії, значно підвищити вірогідність як вхідної, так і залежної від неї вихідної інформації.

Вхідна інформація ІС – це інформація, яка надходить до ІС у вигляді документів, даних, сигналів і потрібна для виконання функцій ІС.

Інформацію реєструють на уніфікованих і неуніфікованих, пристосованих і непристосованих документах для автоматизованої обробки. Неуніфіковані документи мають ряд недоліків:

- послідовність атрибутів у документі не відповідає макету вхідного документа;
- атрибути розкидані по полю документів;
- відсутній ряд атрибутів, які необхідні для ефективної обробки інформації на ЕОМ;
- різноманітність форм документів ускладнює чи виключає їх уніфіковану обробку.

Як наслідок, підвищується трудомісткість і тривалість обробки документів при перенесенні інформації на машинні носії.

В результаті застосування уніфікованих документів, які не мають цих недоліків, підвищується ефективність обробки інформації і перенесення її на машинні носії. Однак ці документи також мають певні недоліки:

- дворазове занесення даних у первинний документ і на машинний носій;
- застосування ручної праці на операціях перенесення інформації на машинний носій.

Документи класифікуються за такими ознаками:

- характером відображення операцій: матеріальні (рух матеріальних цінностей); фінансові (касові й банківські операції); розрахункові;
- місцем складання: внутрішні, зовнішні;
- способом охоплення господарських операцій: разові, накопичувальні, зведені;
- характером заповнення: одно- й багаторядкові, одно- й багатосторонні;
- типізацією: типові (затверджені Міністерством статистики та ін.), індивідуальні;
- цінністю бланків: суворої звітності та ін.;
- способом виготовлення, друкарський, за допомогою технічних засобів та ін.

Як до первісних інформаційних повідомлень, так і до документів висувають такі вимоги.

1. Форми документів мають відповідати вимогам стандартів чи нормативно-технічних документів замовника.
2. Створення повідомлень потрібно обґрунтувати.
3. Форми повідомлень мають бути зручними для сприймання людиною і максимально пристосованими для автоматизованої обробки інформації, яка в них є, містити інформацію у послідовності, яка полегшує заповнення, читання, обробку. Розміри рядків і граф повинні забезпечувати чіткість і зрозумілість усіх записів.
4. Враховувати принцип одноразового введення інформації.
5. Уніфіковані документи мають відповідати вимогам порівняльності атрибутів і показників за змістом і назвою, їх взаємозв'язку при переході з одного рівня управління на інший, при обміні інформацією між різними інформаційними системами і органами управління.
6. Наявність у первісних інформаційних повідомленнях мінімуму атрибутів, достатнього для повного відображення події і отримання результатної інформації, яка виключає невиправдане дублювання інформації і показників, які можливо отримати розрахунком.
7. Форми документів повинні мати стандартні розміри, які передбачені вимогами ГОСТ 9327-60 (формат А3, А4, А5).
8. Атрибути, які переносяться на машинний носій, мають бути виділені потовщеними лініями 0,7 – 1,0 мм, розміщеними в послідовності такій, як і на машинному носієві.
9. Повідомлення мають бути уніфікованими і стандартизованими.
10. При розробці форм документів треба враховувати особливості конкретного друкуючого пристрою.
11. Для кожного розряду атрибута, по можливості, потрібно виділяти спеціальні клітини.
12. Доцільно проектувати документи на різних за кольором бланках.
13. Текстові графи необхідно розміщувати поряд з відповідними їм кодами.

Атрибути мають бути розміщені в такій послідовності: групові й довідкові постійні для документа, групові й довідкові змінні для документа, кількісні.

11.5 МЕТОДИКА ПРОЕКТУВАННЯ ВХІДНИХ ІНФОРМАЦІЙНИХ ПОВІДОМЛЕНЬ

Методика проектування може мати свої особливості залежно від того, розроблюють уніфіковані типові документи чи оригінальні документи, які будуть використовуватись на одному об'єкті.

Для уніфікації документації на основі обстеження документообігу, складу і змісту документів, які застосовуються, виконують такі роботи:

1. Складання і затвердження технічного завдання і методики уніфікації документів.
2. Розробка проектів уніфікованої системи документації різних рівнів.
3. Дослідна перевірка чи дослідна експлуатація проекту системи (проектів форм документів).
4. Розробка остаточної редакції, погодження, затвердження уніфікованої системи інформації (уніфікованих форм документів).

Сама методика проектування документів складається з чотирьох етапів з урахуванням усіх раніше розглянутих вимог.

11.5.1. Встановлення змісту кожного документа (склад атрибутів).

Передбачає визначення складу атрибутів, які використовуються по кожній проблемній сфери, залежно від проблемної сфери, рівня управління, на якому використовуються документи, а також змісту результатної інформації.

Із загального складу атрибутів обирають атрибути конкретної форми первинного повідомлення. При цьому особливу увагу приділяють постійним атрибутам і тим, які переносяться на машинний носій.

11.5.2. Розміщення атрибутів на полі документа за обраною формою побудови.

Поле документа розбивається на зони, призначені для розміщення певних груп атрибутів, що мають, як правило, логічну залежність. Застосовують сполучення різних форм побудови: лінійну і табличну, лінійну та анкетну. Нарешті, дістаємо ескіз форми документа.

11.5.3. Проектування макета машинного носія.

Цей етап наявний, якщо потрібно ув'язати між собою зміст машинного носія і первинного інформаційного повідомлення.

11.5.4. Виготовлення документа, тобто розрахунок бланка документа, уточнення в користувача і затвердження у відповідальних осіб.

Потім виготовлення зразка документа відповідно до вимог стандартів для передачі в друкарню.

Раціональне проектування повідомлень передбачає скорочення кількості атрибутів, їх розміщення в логічній послідовності, зручність для обробки і в користуванні, а також зменшення кількості як самих документів, так і їх кількості.

Є три етапи проходження документації: *до обробки, в процесі обробки і після обробки.* Для кожного з цих етапів потрібно розробити технологію його формування, передавання, обробки і зберігання, аби знати, хто що робить і за що відповідає на кожному з етапів.

ВИКОРИСТАНА ЛІТЕРАТУРА

Для підготовки матеріалу були використані такі джерела інформації:

1. Недашківський О.М.. Планування та проектування інформаційних систем. – Київ, 2014. – 215 с.

12.1 СКЛАДОВІ ЗВ’ЯЗКУ КОРИСТУВАЧ - ПЕОМ

Зв’язок (інтерфейс) – це сукупність засобів і правил, які забезпечують взаємодію між користувачем, ЕОМ і програмами. Можемо виділити три поняття:

- спілкування користувача з комп’ютером;
- спілкування комп’ютера з користувачем;
- подання користувацького зв’язку.

Користувач – комп’ютер. Користувач повинен розпізнати інформацію, що надійшла з комп’ютера, зрозуміти (проаналізувати) її і ввести. Введення реалізується через інтерактивну технологію, елементами якої можуть бути такі дії, як вибір об’єкта за допомогою клавіатури чи

маніпулятора, введення даних. Усе це складає мову дії користувача.

Комп'ютер – користувач. Це спосіб спілкування комп'ютера з користувачем (мова подання), який визначається конкретною прикладною програмною системою, що керує доступом і обробкою інформації і подає її в зрозумілому для користувача вигляді.

Ефективність зв'язку полягає, по-перше, в швидкому, наскільки це можливо, розвитку в користувача простої концептуальної моделі взаємодії. Цього можна досягти через узгодження. Концепція узгодження виходить з того, що при роботі з комп'ютером у користувача формується система очікування однакових реакцій на однакові дії. По-друге, у його конкретності й наочності, що забезпечується застосуванням різноманітних засобів відображення інформації.

Зв'язок може бути узгоджено в трьох аспектах.

Фізична узгодженість належить до технічних засобів: схема клавіатури, використання маніпулятора тощо. Наприклад, для клавіатури фізична узгодженість спостерігається тоді, коли клавіші розміщені в одному й тому самому місці, незалежно від обчислювальної системи.

Синтаксична узгодженість належить до послідовності й порядку появи елементів на екрані та послідовності запитів (наприклад, якщо заголовок завжди розміщується в центрі та верхній частині).

Семантична узгодженість належить до визначення елементів, що складають зв'язок. Наприклад, що означає ВИХІД? Де користувач запитує ВИХІД і що потім відбувається.

Переваги узгодженого зв'язку такі.

Користувачі виграють від того, що знадобиться менше часу для того, щоб навчитися використовувати конкретні прикладні системи, а потім виконувати роботу.

Зменшується кількість помилок користувача, і він почуває себе з системою комфортніше.

Дозволяє виділити загальні модулі зв'язку, стандартизувати його елементи і взаємодію з ними при розробці прикладних систем.

Робота користувача з ПЕОМ має бути зручною і комфортною. На зручність і комфортність впливають такі фактори (табл. 12.1).

Таблиця 12.1 – Фактори, що впливають на зручність і комфорт

Фактори	Вплив	Причина
Соціальні фактори	Психологічний клімат	Емоційний комфорт
Фізична ергономіка	Апаратне забезпечення	Фізичний комфорт
Психологічна ергономіка	Якість розробки програмного забезпеченн	Розумовий комфорт

Якість розробки програмного забезпечення може сприяти успішній роботі користувача.

Ергонометричні характеристики реальної системи можуть суттєво поліпшувати чи погіршувати ставлення до неї користувача:

- конструктивні особливості обладнання;
- якість розробки діалогу;
- доступність і надійність системи;
- чутливість систем.

Для того щоб забезпечити ефективну роботу користувача, необхідно також враховувати його емоційні, психологічні й фізичні особливості. Система може спричинити чи, навпаки, зняти стрес.

Проте всі користувачі мають різний досвід, перед ними стоять різні задачі до системи, висуваються різні вимоги. А особливий інтерес становлять питання організації зв'язку чи діалогового режиму взаємодії людини з ЕОМ, при якому людина і ЕОМ обмінюються даними в темпі, який відповідає темпу обробки даних людиною. Організацію зв'язку і структуру файлів, які використовуються системою, потрібно проектувати нарізно. Діалог між людиною і ЕОМ можемо визначити як обмін інформацією між ЕОМ і користувачем, який відбувається за допомогою інтерактивного терміналу і за певними правилами.

При розробці діалогу необхідно:

- ретельно аналізувати вхідні й вихідні дані;
- знати можливості і мати апаратні й програмні засоби;
- бути послідовним, мати свої прийоми і розробляти «сім'ю» програм, які працюють однаково;
- користуватися прийнятими принципами розробки діалогу;
- «розуміти» задачу і користувача.

До розробки потрібно залучати користувача, передбачаючи засоби адаптації, а також застосовуючи у проектуванні інтерактивний підхід, що веде до розробки дослідних зразків діалогів, з якими працюють користувачі і які змінюються відповідно до їхньої реакції доти, доки не буде створено прийнятний продукт.

Продукт можна оцінити з кількох точок зору:

- простоти освоєння і запам'ятовування операцій системи;
- швидкості досягнення цілей задачі, яка розв'язується за допомогою системи;
- суб'єктивної задоволеності при експлуатації системи.

Так, продукт можна оцінити:

- за контрольним часом, який потрібен певному користувачеві для досягнення потрібного рівня знань;
- за збереженням набутих робочих навичок через деякий час.

Розв'язок задачі можемо оцінити за швидкістю і точністю.

Зв'язок користувача і ЕОМ містить два основних компоненти: процес діалогу, який зв'язує процеси обробки в одну систему; набір процесів введення-виведення, який забезпечує фізичний зв'язок між користувачем і процесом діалогу.

Процес діалогу – це механізм обміну інформацією, який можемо розглядати як оболонку, що охоплює всі процеси, які входять в систему та пов'язані з виконанням певних завдань.

Задачі діалогового процесу:

- визначення завдання, яке користувач покладає на систему;
- прийом логічно пов'язаних вхідних даних від користувача і розміщення їх у змінних відповідного процесу в потрібному форматі;
- виклик процесу виконання необхідного завдання;
- виведення результатів обробки після закінчення процесу у відповідному для користувача форматі.

Головним правилом при цьому є полегшення роботи користувача, а не спрощення процесу обчислень.

У будь-якому діалозі існують різні типи повідомлень (рис. 12.1).

Рисунок 12.1 – Схема побудови коду за ієрархічним методом класифікації

Існує два типи діалогу, який керується системою чи користувачем.

Діалог, що керується системою, це діалог, коли процес жорстко задає, яке завдання можна вибрати і які дані вводити.

Діалог, що керується користувачем, це діалог, коли ініціатива належить користувачеві, він безпосередньо подає команду на виконання необхідного на даному етапі завдання. Більшість операційних систем мають у своєму складі діалоги подібного типу.

Приклад двох діалогів

1. Система: Що потрібно?
2. Користувач: Машина.
3. Система: Волга, Москвич, Жигулі, Мерседес?
4. Користувач: Волга
5. Система: 21, 24, 31
6. Користувач: 31
7. Система: Скільки ? і т. п.
8. Система: Що необхідно?
9. Користувач: Волга, 31, 10 шт.

Діалог, який керується системою, більш зручний, бо він краще підлаштовується під користувача, але при цьому має більше обмежень, ніж діалог, що керується користувачем.

12.2 ПРОЦЕСИ ВВЕДЕННЯ – ВИВЕДЕННЯ

Існують такі найтиповіші пристрої введення – виведення.

12.2.2. Пристрої виведення.

1. Оперативна текстова і графічна інформація (монохромні й кольорові дисплеї).
2. Тверда копія (принтери, графопобудовники).
3. Звукове виведення (синтезатори, звукогенератори).
4. Фактографічне виведення (інтерактивна інформація).

12.2.2. Пристрої введення.

1. Введення даних людиною:

- клавіатура (текстове введення);
- планшети, сканери (графічне введення).

2. Автоматизоване збирання інформації:

- засоби зчитування документів;
- звуковий і рядковий сканери.

3. Позиціювання і вибір:

- світлове перо;
- сенсорний екран;
- маніпулятор «миша», «джойстик» і куля.

12.2.3. Фактори вибору пристроїв.

Існують такі фактори вибору пристроїв введення-виведення інформації:

- зміст і формат оброблюваних даних;
- обсяг уведення – виведення;
- обмеження, висунуті користувачем і робочим середовищем;
- обмеження, пов'язані з іншими апаратними і програмними засобами.

Клавіатура – це основний пристрій введення даних в ЕОМ при інтерактивному режимі. Вона використовується для введення малих і середніх обсягів символьних даних залежно від досвіду користувача. Для великих обсягів даних слід застосовувати оптичні й магнітні зчитуючі пристрої (зчитувачі кодів, пристрої для зчитування штрихових кодів, кредитні картки, спеціальна клавіатура, сканери тощо).

12.2.4. Класифікація основних процесів введення – виведення.

1. Уведення текстового повідомлення:

- з використанням стандартних процедур введення;
- у режимі посимвольного введення;
- з використанням спеціальних символів.

2. Уведення повідомлення типу «ВКАЗАТИ» чи «ВИБРАТИ»:

- перегляд запропонованого списку операцій і вибір потрібної;
- вибір даних з будь-якого місця екрана.

3. Введення графічного повідомлення.
4. Виведення текстового повідомлення:

- у поточну позицію на пристрій;
- у задану позицію на пристрій;
- з визначенням конкретного формату відображення.

5. Виведення графічного повідомлення.

12.2.5. Параметри повідомлення.

Будь-який текст повідомлення можемо описати такими параметрами: *ЩО? ДЕ? ЯК?*

ЩО потрібно вивести, тобто рядок символів, що містить склад повідомлення.

ДЕ текст має бути розміщений, тобто позиція на пристрої виведення.

ЯК текст повинен бути виведений, тобто список атрибутів, що визначають формат даних, які виводяться, а також додаткові допоміжні параметри (колір, світлова пульсація, підвищена яскравість і т. ін.).

Формат, згідно з яким користувач вводить свої повідомлення, називається граматикою діалогу. Існує кілька варіантів граматики діалогу:

- коди;
- ланцюжок ключових слів, які нагадують програму;
- обмежена українська, російська, англійська та інші мови;
- природна українська, російська, англійська та інші мови.

Інформацію, що виводиться на екран, можемо поділити на окремі об’єкти з такими характеристиками (словник полів):

- зміст, що описує об’єкт. Кожний об’єкт повинен мати однозначне визначення, щоб користувач знав, до чого він звертається;
- область (поле), в межах якого об’єкт відображається на екрані (рядок, стовпець, ширина);
- множина атрибутів, що описують даний об’єкт (передній план, тло, контрастність і т.п.).

Область екрана, в якій розміщений об’єкт, має бути чітко виділена, а число об’єктів має бути незначним, аби користувач не розгублювався при їх ідентифікації (рис. 12.2).

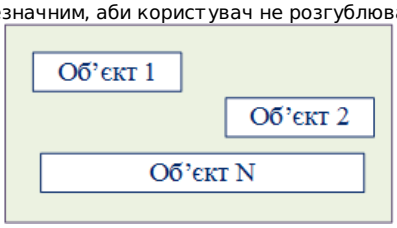


Рисунок 12.2 – Розміщення об’єктів на екрані

12.2.6. Способи вибору.

Існують два способи вибору: абсолютний і відносний.

Абсолютний дозволяє користувачеві вказати будь-яке місце на екрані, незалежно від того, розміщений там об’єкт чи ні. Після введення можна отримати від пристрою введення точні координати. Цей спосіб використовується під час роботи зі спеціальними пристроями введення даних (наприклад, маніпулятор чи сенсорний екран).

При відносному способі вибору межі переміщення по екрану обмежені лише списком символів.

Однак у кожному разі користувач потребує ясного і оперативного підтвердження того, на що він вказує в даний момент. Також потрібно визначити засіб завершення процесу введення. Це або призначення якоїсь клавіші, або процес уведення завершується, коли код цього об’єкта вводиться повністю

12.3 ДІАЛОГ

Виділяють чотири основні структури типів діалогу: запитання і відповідь; меню; екранних форм; на базі команд.

12.3.1. Критерії оцінки придатності діалогу.

Придатність структури до діалогу можна оцінювати за такими основними критеріями: природністю; послідовністю; стислістю (короткий); підтримкою користувача; гнучкістю.

Природним діалогом є такий, який не змушує користувача, котрий взаємодіє із системою, суттєво змінювати свої традиційні прийоми роботи. Тому діалог має вестися державною мовою,

розмовним стилем, а не письмовим; окрім того, слід уникати як надмірної пишномовності, так і «фамільярності».
Наприклад: коротка підказка у вигляді: ВАРІАНТ? більш інформативна і більш зручна, ніж чемність фрази: МОРОЗ, БУДЬ ЛАСКА, ВИБЕРІТЬ ПОТРІБНИЙ ВАРІАНТ.
Фрази, по можливості, не повинні потребувати додаткових пояснень
краще «ДРУК» «КОПІЮВАННЯ», ніж «PRINT», «COPY»
Жаргон припустимий лише за умови, що він використовується в середовищі користувача.
Наприклад: не «КОРЕКЦІЯ ІСНУЮЧОГО ФАЙЛА», а «ОБРОБКА ВИМОГ»
Порядок запиту має бути таким, у якому користувач звичайно обробляє інформацію.

Неприродний діалог часто є наслідком того, що розробник неознайомлений з тими прийомами, якими користувач зазвичай розв'язує задачу без підтримки системи. Тому слід розробити і випробувати дослідний зразок системи.

Наприклад, додаткову роботу користувача з підготовки даних перед введенням-виведенням видно, коли він у процесі роботи за терміналом буде користуватися олівцем і папером.

Діалог, який відрізняється *логічною послідовністю*, гарантує, що користувач, котрий освоїв одну частину системи, легко розбереться з особливостями іншої частини системи:

- послідовність у побудові фраз передбачає, що коди, які вводяться, наприклад ключові слова, завжди трактуються однаково. Наприклад: допомогу від системи користувач отримає, натиснувши клавішу F1, якщо виникне будь-яке інше питання, він також повинен натиснути клавішу F1, тобто ця клавіша не може мати інших функцій, крім виклику довідкової інформації.
- послідовність у використанні формату даних означає, що аналогічні поля завжди будуть подані системою в одному і тому самому форматі.
- послідовність у розміщенні даних на екрані в різних ситуаціях, схожих за функціями, які реалізуються, є гарантією того, що користувачеві відомо, де шукати на екрані інструкцію, повідомлення про помилку і т. ін.

Стислий діалог потребує від користувача введення лише мінімуму інформації, яка необхідна для роботи системи. І тому:

- чим меншою буде кількість потрібних натискань на клавішу, тим швидше відбувається діалог із меншою кількістю помилок;
- у діалозі не слід вимагати інформацію, яку можна сформувати автоматично, наприклад: за кодом назву чи прізвище з бази даних, чи інформацію, яка введена раніше (наприклад, поточну дату);
- вихідні повідомлення повинні містити лише ту інформацію, яка потрібна користувачеві, у вигляді, прийнятному для сприйняття, із залученням мінімуму засобів для виділення частини інформації.

Підтримка користувача в процесі діалогу – це та допомога, яку користувач отримує від діалогу під час його роботи із системою. Три основні частини підтримки користувача:

- кількість і якість наявних інструкцій;
- характер повідомлень про помилки, які видаються;
- підтвердження яких-небудь дій системи.

Інструкції для користувача виводяться у вигляді підказок чи довідкової інформації. Характер і кількість інструкцій мають відповідати досвіду роботи користувача із системою і його намірам. *Довідкова інформація* має з'являтися тоді, коли вона потрібна, і в доступній формі.

Повідомлення про помилки мають точно пояснювати, в чому помилка і які дії потрібно зробити, аби її виправити, а не обмежуватися загальними фразами «синтаксична помилка» чи таємничими кодами типу «ОС1».

Повідомлення, що підтверджують які-небудь дії системи, потрібні для того, щоб користувач міг переконатися в тому, що система виконує, виконала чи виконуватиме необхідну дію.

Наприклад: підтвердження дії є необхідним, коли можуть виконатися незворотні дії (видалення запису із файлу); коли вводиться код і потрібно підтвердити, вибрано той запис чи ні (табельний номер працівника і підтвердження правильності прізвища).

Гнучкий діалог – це міра того, наскільки добре він відповідає рівню підготовки і продуктивності роботи користувача. Так, гнучкість передбачає, що діалог може змінювати свою структуру чи вхідні дані.

12.3.2. Структури діалогу.

Усі чотири структури діалогу різною мірою відповідають переліченим критеріям і забезпечують різний рівень підтримки користувача.

Структура діалогу типу «запит – відповідь» ґрунтується на аналогії зі звичайним інтерв'ю:
ЯКА ОПЕРАЦІЯ? – ПОСТАВКА
КОМАНДА? – ВІДІСЛАТИ
ТИП ОБ'ЄКТА? – РАХУНОК
НОМЕР РАХУНКА? – 12543
КЛІЄНТ? – МП «ПРОГРЕС»

Існують системи, де відповіді дають природною мовою, але більше використовують речення із одного слова з обмеженою граматикою.

Для полегшення сприйняття довжину повідомлення потрібно обмежити приблизно 40 символами, які виводяться в лівій частині екрана, становлячи 2/3 ширини екрана; крім того, запитання системи мають відрізнятися від відповідей користувача (наприклад, зміна регістру, колір відповідей, контрастність, інверсія, розділові знаки).

У цій структурі виділяють три основних кроки: виведення запитання, введення відповіді та контроль вірогідності відповіді.

Структура типу «запитання – відповідь» – це розумний компроміс для різних рівнів підготовки користувачів. Вона використовується в таких випадках:

- діапазон вхідних величин надто великий для структури типу «меню» чи надто складний для структури на основі мови команд;
- наступне запитання залежить від відповіді на попереднє, діалог має багато відгалужень і на кожне запитання передбачається багато відповідей.
- часто використовується в експертних системах.

Структура діалогу типу «меню» відображає точний список варіантів і дає можливість користувачеві вибрати один із них у такий спосіб:

- введенням ідентифікатора з клавіатури;
- введенням мнемонічних кодів;
- переглядання списку на екрані з наступним вибором;
- прямою вказівкою на екрані.

Меню може мати вигляд блоку даних, рядка даних, піктограм; спливаючого (додаткового) меню, яке з'являється в процесі переміщення курсора по екрану з меню верхнього рівня (рис. 12.3).

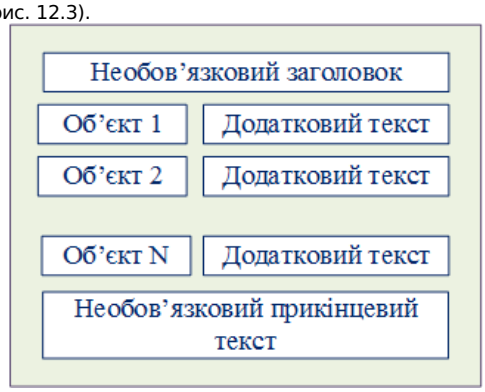


Рисунок 12.3 – Структура діалогу типу «меню»

Кожне меню може мати необов'язковий заголовок, необов'язковий прикінцевий текст, і основний текст меню, який складається зі списку об'єктів вибору і необов'язкового додаткового тексту, що описує якості кожного пункту.

Структура типу «меню» використовується там, де:

- діапазон можливих відповідей досить невеликий, і всі вони можуть бути явно відображені;
- користувач повинен бачити всі можливі варіанти відповідей.

У структурі діалогу типу «екранних форм» перед користувачем ставиться одразу кілька запитань і відповідь на попереднє не впливає на те, яким буде наступне. Подібні форми широко використовуються при замовленні товарів, білетів, складанні анкет, заповненні форми вхідного повідомлення тощо. Форми заповнюють зліва направо і згори вниз. Можна вибрати послідовність заповнення форми і працювати доти, доки не буде прийнято рішення про вірогідність даних і закінчено її формування. Система може перевіряти кожну відповідь після введення чи виводити список помилок після заповнення всієї форми. Ця структура будується в два етапи:

- форма відображується повністю;
- питання повторюються доти, доки не закінчиться заповнення форми.

Введення відповіді на запитання можна завершити двома способами:

- при явному завершенні користувачеві потрібно ввести для кожного поля відповіді повний символ завершення.
- при автоматичному завершенні перехід до наступного виконується одразу, тільки-но буде заповненим поле для введення відповіді.

Структура типу екранних форм зручна там, де можемо передбачити стандартну послідовність уведення даних.

Структура діалогу типу «мови команд» будується на основі мови програмування чи операційної системи. Вона широко використовується так, як і тип «меню», але особливо в операційних системах. Відповідальність за достовірність команд, які подаються, покладено на користувача. Вона забезпечує ширші можливості вибору в будь-якому місці діалогу і не потребує ієрархічної організації фонових завдань. Розробник повинен уникати надмірної функціональності, виходячи з бажання створити свій командний рядок для кожної функції.

Структура типу мови команд використовується там, де:

- кількість значень для введення невелика і їх можна запам'ятати;
- обмежена кількість відповідей достатня для того, щоб ідентифікувати як необхідну задачу, так і необхідні дані.

Усі чотири структури мало різняться, є різновидом структури типу «запитання - відповідь» і застосовуються залежно від специфіки задачі, вимог користувача і можливостей обчислювальної техніки.

Діалог для всієї системи важко побудувати, використовуючи лише один тип діалогу. Для різних частин діалогу залежно від їх конкретних характеристик вибирають найбільш придатний.

12.4 РОЗМІЩЕННЯ ДАНИХ НА ЕКРАНІ ДИСПЛЕЯ

Зовнішній вигляд екрана залежить від того, які поля повідомлень відображаються, в якому місці і з якими атрибутами.

Ітеративний процес розміщення даних на екрані складається з таких етапів:

- вирішити, яка інформація, тобто які поля мають з'являтися на екрані;
- визначити головний формат цієї інформації;
- вирішити, де вона має з'являтися на екрані, тобто визначити область виведення для кожного поля;
- вирішити, які засоби потрібні для виділення полів, тобто які атрибути необхідні для кожного поля;
- розробити проект розміщення даних на екрані;
- оцінити ефективність цього розміщення.

Цей процес повторюється доти, доки користувач не буде задоволений.

Інформацію потрібно розміщувати так, аби користувач міг переглядати екран у логічній послідовності й легко виводити потрібну інформацію, ідентифікувати зв'язані групи інформації, розрізняти виняткові ситуації (повідомлення про помилки чи попередження), а також визначати, які дії з його боку потрібні (чи потрібні взагалі) для продовження виконання завдання.

На екрані має розміщуватись лише та інформація, яка дійсно потрібна користувачеві на даному етапі роботи.

1-й етап. На основі вивчення проблемної сфери визначають, яка інформація потрібна користувачеві і яка в даний момент має розміщуватися на екрані (зміст відеокадра).

2-й етап. Розробник повинен визначити розмір областей виведення і атрибути, які пов'язані з кожним полем.

Поля вхідних і вихідних даних повинні мати назву, яка точно визначає зміст відповідного поля, і відокремлюватися від даних. Основні принципи вибору назви поля:

Назви мають бути короткими, проте скорочення не повинні бути довільними.

Для відокремлення назви поля від їх значень останні виділяються за допомогою таких засобів: розділових знаків; дужок; великих літер; підвищеної яскравості; інверсії; кольору.

Назву потрібно розміщувати у природному і логічному зв'язку з відповідними значеннями полів або на тому самому рядку і зліва для одного значення поля чи у вигляді заголовків над відповідними полями даних.

3-й етап. Визначають місце раціонального розподілу інформації на екрані. Інформація може розміщуватися в ієрархічній послідовності: екран, відеокадр, вікно, панель, поле.

Відеокадр – це відображення інформації в закінченій логічній відповідності на конкретний момент часу або сформоване зображення для одночасного зображення інформації на екрані. На екрані чи в окремих його частинах може розкриватися вікно для тієї чи іншої функції. Використовуючи кілька вікон, користувач може одночасно спостерігати за кількома панелями чи процесами. Існує три типи вікон:

- первинне – це вікно, де починається діалог;
- вторинне – викликається з первинних вікон;
- спливаюче – частина екрана, де виведена інформація розширює діалог користувача з первинним чи вторинним вікном.

Первинні й вторинні вікна мають заголовки у верхній частині.

На екрані і у вікнах зібрана інформація за будь-якими ознаками може розміщуватися у вигляді панелей. До основних елементів панелі відносять (рис. 12.4): елементи, що розділяють сфери; ідентифікатор панелі; заголовок панелі; інструкцію; вказівки протяжки (скролінгу); область повідомлень; область команд; область функціональних клавіш.

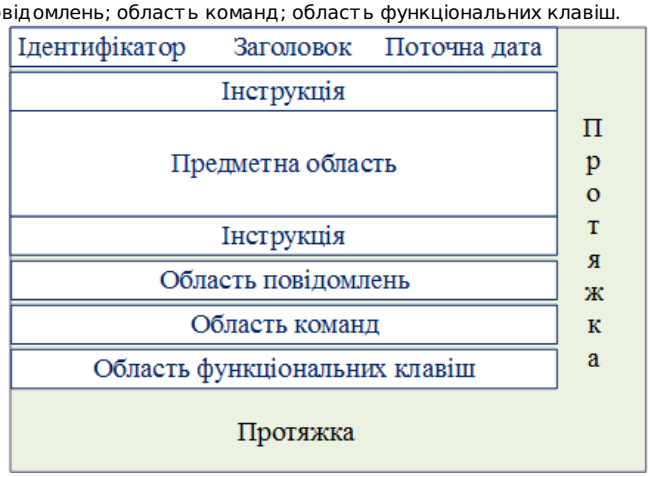


Рисунок 12.4 – Шаблон панелі

Елементами, що відокремлюють тіло панелі від сусідніх областей, можуть бути кольорові межі, лінії, вільні рядки або заголовки стовпчиків.

Ідентифікатор панелі містить захищену інформацію і є способом ідентифікації панелі у діалозі. Він розміщується в першому рядку тіла панелі, вирівняний вліво і відокремлений від заголовка панелі. Він унікальний і може показувати місце в ієрархічній системі меню.

Заголовок панелі, розміщений у верхній частині панелі, повідомляє користувачеві, яка інформація міститься в тілі панелі. Центр має збігатись з центром вікна, навіть якщо розміри вікна будуть змінюватися. Заголовок панелі має складатися з одного рядка. У правому кутку може розміщуватися *поточна дата* і час, а також підтвердження, що система працює.

Інструкція повідомляє користувачеві, що треба зробити на панелі і як продовжити роботу. Інструкція може бути розміщена у верхній та нижній частинах тіла панелі нижче заголовка панелі. *Верхня інструкція* має відокремлюватися від меню порожнім рядком, а *нижня* може бути розміщена безпосередньо над областю повідомлень чи команд в одному чи кількох рядках панелі. Текст вирівнюється вліво. У верхній інструкції найчастіше виводять підказки, які визначають спосіб роботи з інформацією, у нижній – повідомлення про те, що робити далі.

Вказівка *протяжки* повідомляє, що існує більше інформації, ніж відображено в частині панелі. Ця інформація може бути в трьох формах: текстовій; стрілочками; лінійкою.

Область повідомлень. Їх рекомендується зображувати у спливаючих вікнах чи зарезервувати кілька рядків у нижній частині панелі. Вона розміщується над областю команд, яка має знаходитися в найнижчій частині панелі. Вона також повинна відокремлюватися від області команд або області функціональних клавіш (якщо вони відображені) роздільником. Це довідкові, критичні, попереджувальні повідомлення, а також повідомлення про помилки.

Область введення команд може бути розміщена в основній панелі у вторинному чи спливаючому вікні.

В *області функціональних клавіш* описують можливості, які доступні користувачеві у поточному стані прикладної системи. Вона розміщується в нижній частині панелі. Для кожної панелі потрібно визначити її область функціональних клавіш.

Компактність і місце розміщення інформації поняття суб'єктивні, які залежать від конкретного користувача і специфіки задачі, однак можна визначити загальні принципи:

- залишати порожньою приблизно половину екрана;

- залишати порожнім рядок після кожного п’ятого рядка таблиці;
- залишати 4 чи 5 пропусків між стовпчиками таблиці;
- фрагменти тексту потрібно розміщувати на екрані так, аби погляд користувача сам перемішувався по екрану в потрібному напрямку;
- зміст полів має розміщуватися і вирівнюватися біля горизонтальних і вертикальних осей, а не притискатися до країв екрана;
- початок розміщення – з лівого верхнього кутка; переміщувати потрібно зліва направо і згори вниз. Необхідно враховувати естетичні характеристики. Легше слідкувати за даними, які правильно розміщені на екрані, при цьому підвищується безпомилкова робота. Правило: треба уникати того, щоб користувач, заповнюючи екранну форму, витрачав стільки ж зусиль, як і розробник при її розробці.

4-й етап. Визначають атрибути, які привертають увагу користувача до деякої частини екрана чи дії. До цих атрибутів поля відносять: колір символів; колір тла; рівень яскравості; режим мерехтіння; звук.

Кожним із цих ефектів можна досягти іншого ефекту.

Тло мерехтіння – найсильніший засіб, проте він також відволікає. Краще обмежитися однією позицією символу у виділеному полі.

Колір – це другий визначальний фактор, який привертає увагу.

Правила використання кольорів такі:

- використовуйте мінімальну кількість кольорів, щонайбільше три чи чотири на одному екрані;
- для великих панелей використовуйте колір тла;
- добирайте яскраві кольори для виділення даних, а спокійніші тони – для тла;
- для виділення двох областей для однієї беріть чорний колір чи колір з одного кінця спектра, а для іншої – білий колір чи колір із середини спектра;
- колір потрібно використовувати виходячи з уявлень про нього користувача;
- поекспериментуйте з різними відтінками на реальному екрані;
- різні кольори сприймаються по-різному. Область, тло якої зображено більш теплими відтінками у червоній частині спектра, виглядає крупнішим, ніж область, колір якої перебуває в голубій частині;
- область екрана на білому тлі чи на тлі в середині спектра, видається яскравішою й легше сприймається при різному освітленні. Використання різної яскравості – найменш надокучливий спосіб привертання уваги.

Можна виділяти підкресленням чи іншим шрифтом. Для того щоб привертати увагу, можна використовувати звук.

5-й етап. Розробка проекту екранної форми. Його можна спочатку спроектувати на папері, а потім розробити на екрані, роздрукувати і затвердити у замовника.

6-й етап. Оцінка якості розробки екранних форм є доволі важкою роботою, оскільки зміст інформації впливає на людину і потрібно відокремити його від форми. Для цього використовують два методи: прямокутників і виділених точок.

Метод прямокутників ділить екран на частини. Кожна частина екрана заповнюється текстом і відокремлюється від інших щонайменше одним пропуском по всьому периметру. В результаті екран розбивається на групи прямокутників тексту. Через центр екрана проводять вісь, яка дозволяє оцінити збалансованість даних. За кількістю і розміром прямокутників можна оцінити характер розміщення інформації. Велика кількість маленьких прямокутників має безладний вигляд.

Метод виділених точок дозволяє визначити область екрана, до якого буде привернута увага користувача через інший рівень яскравості в цьому місці. Оцінюють їх симетричність відносно центральних осей. Однак ці методи не можуть дати кількісної оцінки.

12.5 ПІДТРИМКА КОРИСТУВАЧА

Підтримка користувача з боку системи спрямована на досягнення трьох головних цілей:

- ввести користувача в курс роботи системи, пояснити її можливості та основні принципи;
- забезпечити інструкцією, яка дасть змогу розв’язати на машині поставлену задачу;
- допомогти користувачеві в роботі і забезпечити його довідковою інформацією.

12.5.1. Складові підтримки користувача.

Основні складові підтримки користувача – внутрішня довідкова інформація і обробка помилок, які складають дружній інтерфейс, не виключають і оформлення зовнішніх документів – інструкцію з використання і технічних описів. На систему можна скласти весь спектр документів, якого вимагають державні стандарти, чи їх частину за погодженням із замовником, але необхідними є такі:

- загальний огляд (постановка задач), у якому описують призначення системи, основні поняття проблемної області;
- керівництво системного програміста – завантаження системи і БД;
- інструкція з використання, в якій наведено дані для початку роботи, як увімкнути апаратуру і т.ін.;
- система навчання – створена окремо чи розміщена в самій системі. Але доцільно мати спеціальну навчальну версію системи;
- система має реалізовувати обробку помилок і видачу довідкової інформації.

12.5.2. Типи помилок.

Помилки бувають трьох типів:

1. Система виходить із ладу.
2. Система виявляє неможливість подальшої обробки отриманих нею даних і просить користувача скоригувати їх.
3. Система виконує роботу, але результат відрізняється від того, на який сподівався користувач, наприклад нова версія файлу знищується старою, можливо, користувач неправильно скористався командою копіювання.

Усі системи потребують механізму перевірки коректності вхідних даних для припинення помилкового введення і повідомлення користувача про це. Повідомлення про помилку призначається для користувача, а не для програміста; воно має бути інформативним для користувача. В ідеальному випадку повідомлення повинно точно визначати причину помилки.

Перевірка може виконуватись на формат (розмір, буквений чи цифровий, дата, ім'я), суперечливість даних (дата), несумісність з іншими вхідними даними, наприклад, рівність загальної суми сумі окремих платежів; а також тоді, коли запитуються неіснуючі елементи, наприклад, для введеного коду замовника у файлі «Замовник» немає запису.

Довідкова інформація і повідомлення про помилки мають багато спільного. Вона повинна бути інформативною і зрозумілою користувачеві.

Допомога має бути своєчасною і доступною з будь-якої частини системи без винятку. Довідкова інформація може бути багаторівневою, що дає змогу викликати більш детальну інформацію.

ВИКОРИСТАНА ЛІТЕРАТУРА

Для підготовки матеріалу були використані такі джерела інформації:

1. Недашківський О.М.. Планування та проектування інформаційних систем. – Київ, 2014. – 215 с.