Проектування інформаційних систем

Марченко Анна Вікторівна

2015

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.

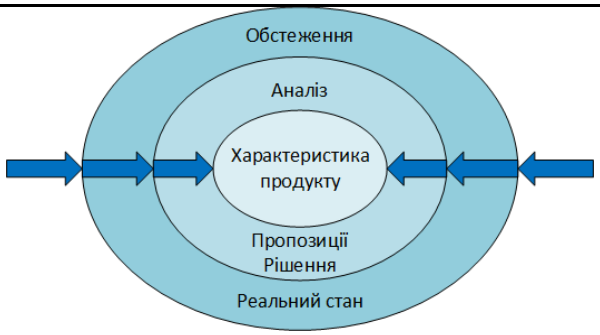


Рисунок 1.1 – Прийняття рішення про доцільність створення ІС

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.

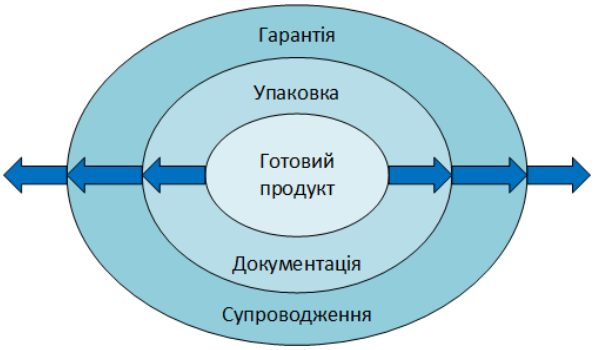


Рисунок 1.2 – Створення ІС

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).

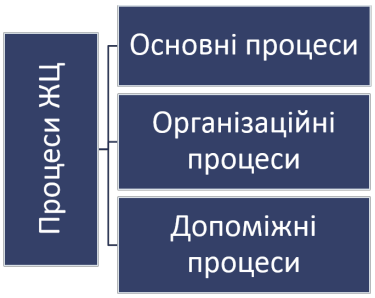


Рисунок 1.3 – Процеси життєвого циклу розроблення ПЗ

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.

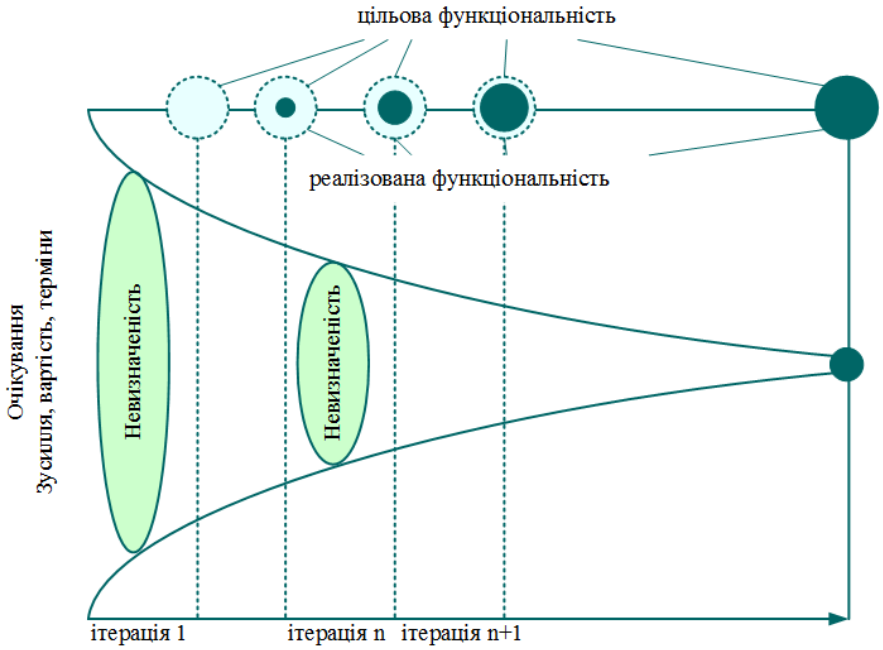


Рисунок 1.6 – Зниження невизначеності та інкрементальне розширення функціональності при ітеративній організації життєвого циклу

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.

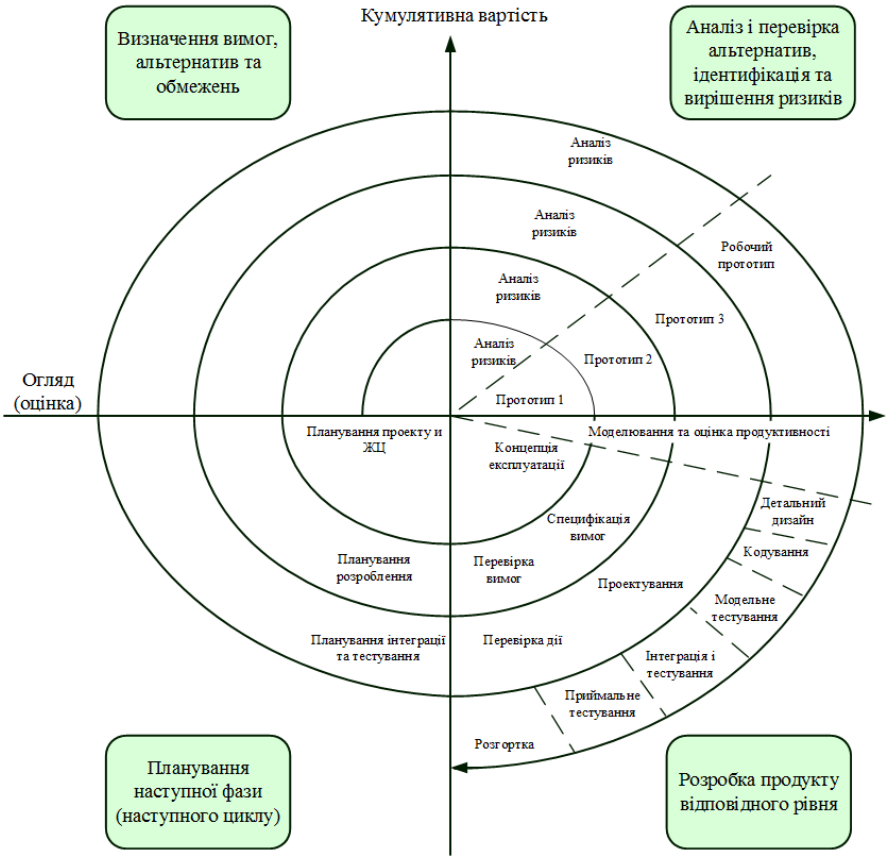


Рисунок 1.7 – Оригінальна спіральна модель ЖЦ, розроблена Боемом

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).

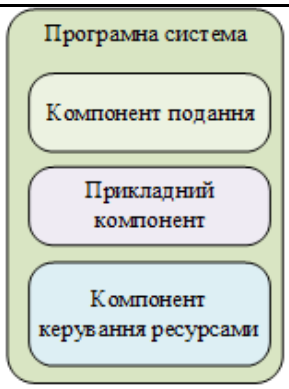


Рисунок 2.2 – Компоненти програмної системи

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.

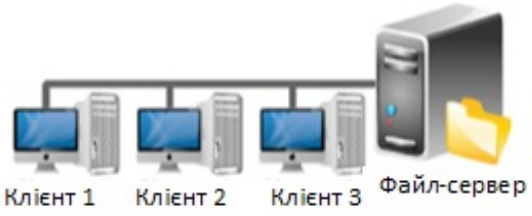


Рисунок 2.3 – Подання «файл-серверної» архітектури

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.

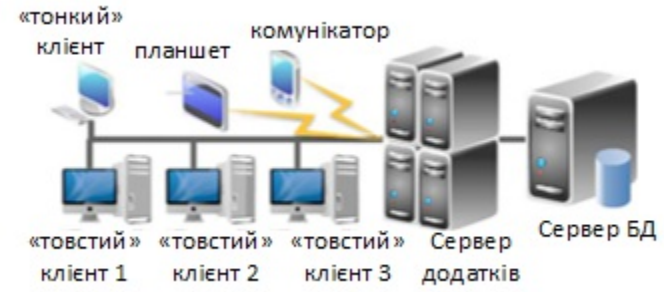


Рисунок 2.5 – Приклад багатоланкової «клієнт-серверної» архітектури

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.

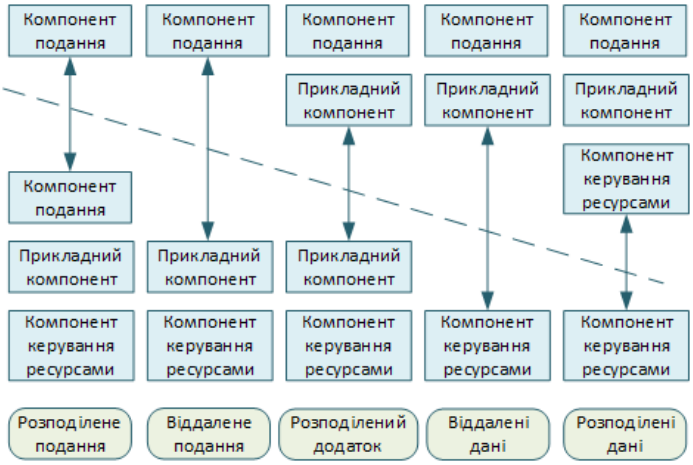


Рисунок 2.6 – Приклади розподілу функціональних компонентів

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).

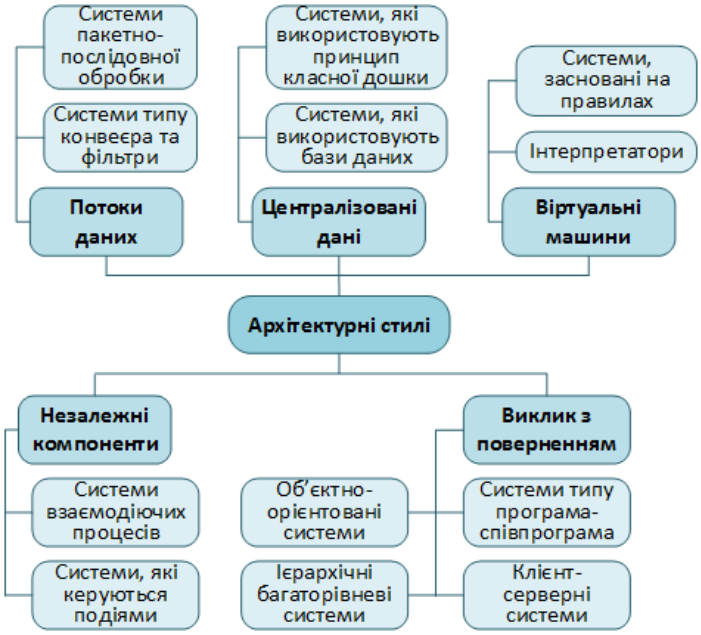


Рисунок 2.7 – Класифікація архітектурних стилів

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.

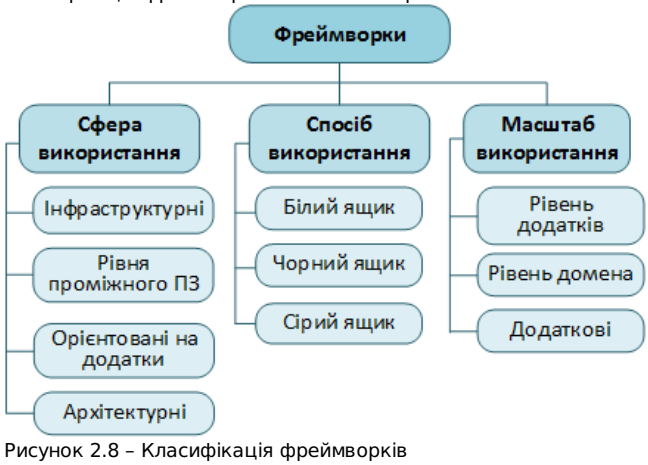


Рисунок 2.8 – Класифікація фреймворків

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.

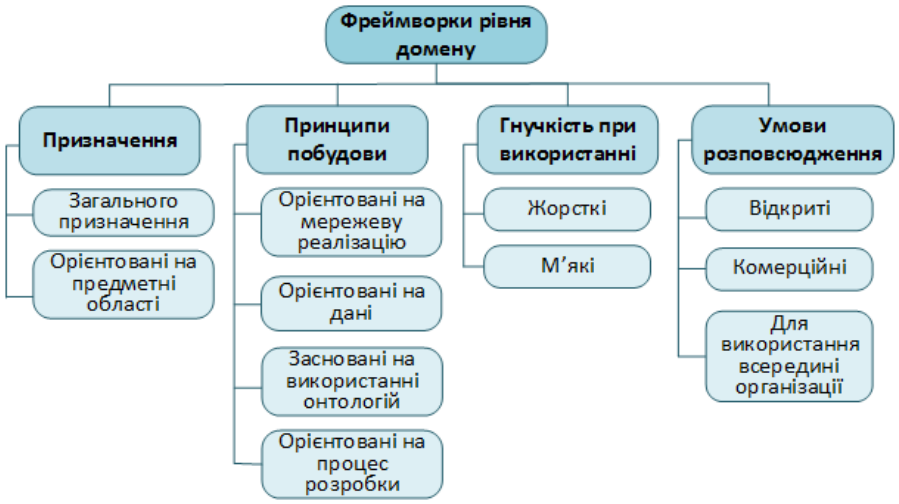


Рисунок 2.10 – Класифікація фреймворків рівня домену

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.

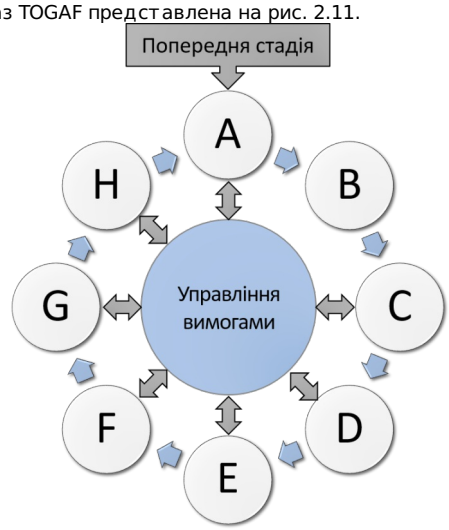


Рисунок 2.11 – Схема фаз відповідно методиці ADM

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.

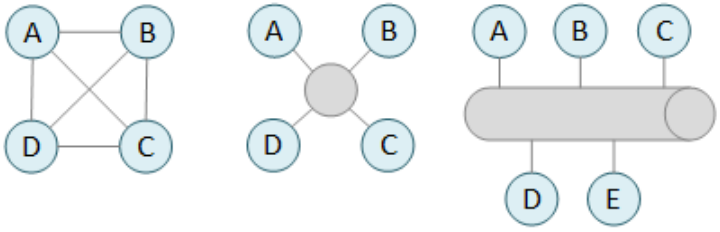
Існують три альтернативних топології інтеграції:

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.

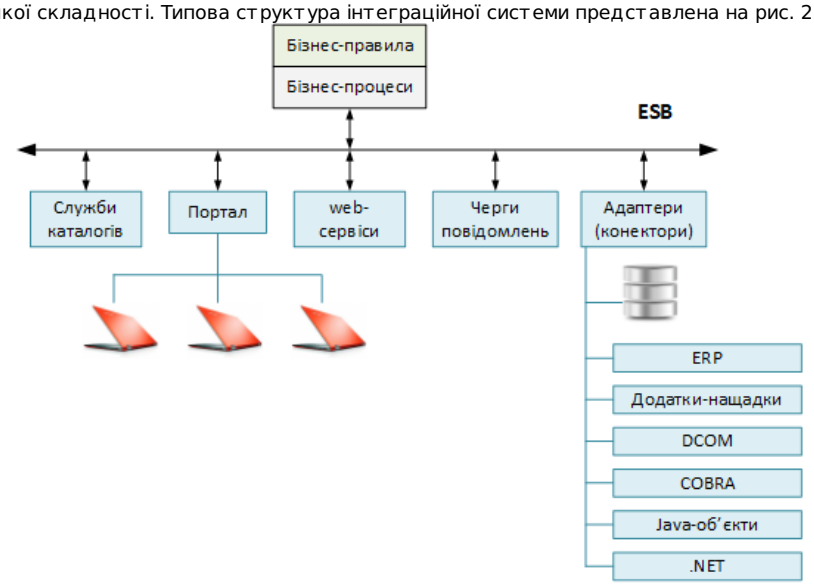


Рисунок 2.13 – Типова структура інтеграційної системи

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.

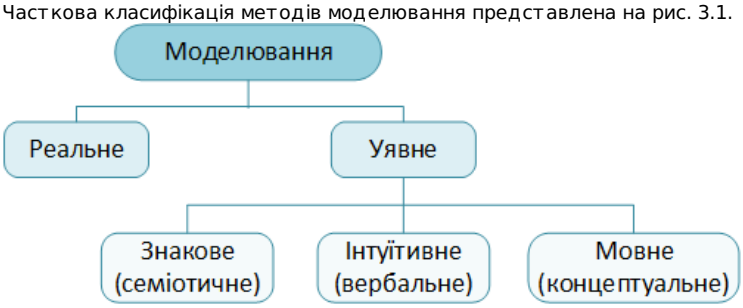


Рисунок 3.1 – Класифікація методів моделювання

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.

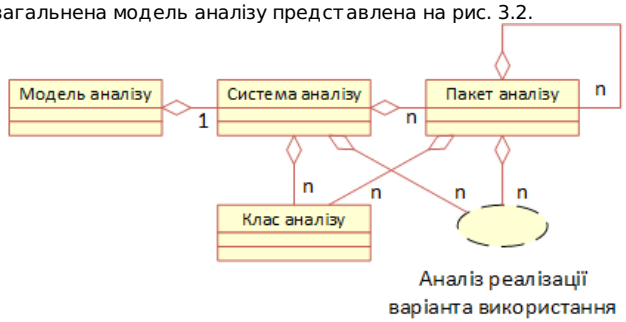


Рисунок 3.2 –Узагальнена модель аналізу

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.

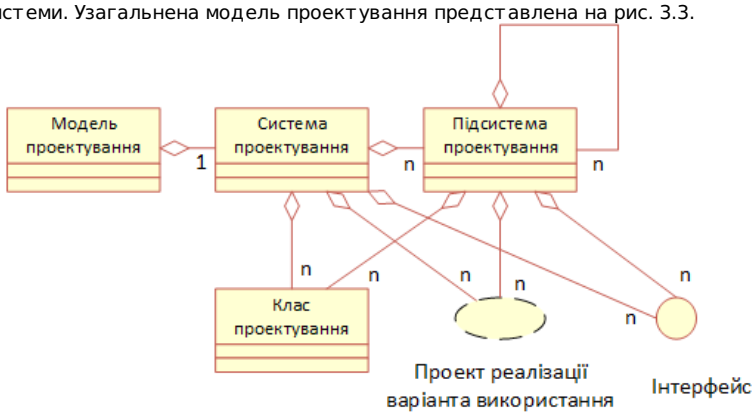


Рисунок 3.3 – Узагальнена модель проектування

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.