МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ

Балтовський О.О., Форос Г.В., Сіфоров О.І.

ТЕОРІЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

Навчальний посібник

Одеса 2024

УДК 004.4 (075.8)
ББК 32.973.2-018я73-1
А21
Рекомендовано до друку науково-методичною радою Одеського державного університету внутрішніх справ
Авторський колектив:
Балтовський О.О. – доктор технічних наук, доцент, професор кафедри кібербезпеки та інформаційного забезпечення ОДУВС;
Форос Г.В. – кандидат юридичних наук, доцент, доцент кафедри кібербезпеки та інформаційного забезпечення ОДУВС;
Сіфоров О.М. – кандидат технічних наук, доцент, доцент кафедри кібербезпеки та інформаційного забезпечення ОДУВС.

Рецензенти:
Наталія Логінова – кандидат педагогічних наук, доцент, завідувачка кафедри інформаційних технологій Національного університету «Юридична академія»
Афонін Дмитро - кандидат юридичних наук, доцент, завідувач науково-дослідної лабораторії з актуальних питань кримінального аналізу навчально-наукового інституту підготовки фахівців для підрозділів кримінальної поліції

Балтовський О.О., Форос Г.В, Сіфоров О.І. Теорія та проектування інформаційних систем / За заг. ред. д.т.н., доц. О.А. Балтовського. Одеський держ. унів-т внутр. справ. 2024. 143 с.

Навчальний посібник "Теорія та проектування інформаційних систем" охоплює основні принципи, методи та інструменти, необхідні для розробки ефективних інформаційних систем. Посібник розглядає концептуальні основи інформаційних систем, включаючи їх архітектуру, моделювання даних, аналіз вимог, проектування інтерфейсів користувача та управління проєктами. Особлива увага приділяється сучасним технологіям та підходам, таким як об'єктно-орієнтоване проектування, реляційні бази даних, клієнт-серверні архітектури та засоби для забезпечення безпеки даних. Посібник також містить прикл ади реальних проєктів та практичні завдання, що сприяють глибокому розумінню та застосуванню теоретичних знань у практичній діяльності. Він призначений для студентів вищих навчальних закладів, що вивчають інформаційні технології, а також для фахівців, що прагнуть поглибити свої знання у сфері проектування інформаційних систем.
Навчальний посібник призначений для здобувачів, які навчаються за спеціальністю124 « Системний аналіз» з дисципліни «Теорія та проектування інформаційних систем». Матеріал посібника може використовуватися і в практичній діяльності студентів інших спеціальностй і фахівців, що займаються розробкою і впровадженням інформаційних систем.

ЗМІСТ

ВСТУП

Розробка інформаційної системи (ІС) – від початкової стадії до впровадження – складається з трьох основних послідовних етапів: аналізу, проектування та реалізації. У цьому посібнику детально розглянуто методи і прийоми, що застосовуються на етапах аналізу та проектування. Питання, пов'язані з реалізацією, включаючи приклади програм, розглядаються лише в тій мірі, в якій вони необхідні для етапу проектування.

Проектування інформаційних систем є логічно складним, трудомістким і тривалим процесом, що вимагає високої кваліфікації спеціалістів, які беруть участь у цьому процесі. Проте на сьогоднішній день проектування інформаційних систем часто виконується на інтуїтивному рівні, використовуючи неформалізовані методи, що включають елементи мистецтва, практичний досвід, експертні оцінки та дорогі експериментальні перевірки якості функціонування інформаційних систем. Додатково, у процесі створення та експлуатації інформаційних систем інформаційні потреби користувачів постійно змінюються або уточнюються, що ще більше ускладнює розробку та підтримку таких систем.

Основна частина трудових витрат при створенні ІС припадає на розробку прикладного програмного забезпечення (ПЗ) та баз даних (БД). Сьогодні виробництво ПЗ є однією з найбільших галузей світової економіки, в якій задіяні близько трьох мільйонів спеціалістів, включаючи програмістів та розробників ПЗ.

На початку 70-х років у США спостерігалася криза програмування (software crisis). Це проявлялося в тому, що великі проекти виконувалися із затримками або з перевищенням кошторису, розроблений продукт не мав необхідних функціональних можливостей, його продуктивність була низькою, а якість програмного забезпечення не відповідала очікуванням користувачів.

Причини можливих невдач включають нечітке та неповне формулювання вимог до ПЗ, недостатнє залучення користувачів у роботу над проектом, відсутність необхідних ресурсів, незадовільне планування, часті зміни вимог та специфікацій, новизну використовуваної технології для організації, відсутність грамотного управління проектом та недостатню підтримку з боку вищого керівництва.

Потреба контролювати процес розробки ПЗ, прогнозувати та гарантувати вартість розробки, терміни та якість результатів призвела в кінці 70-х років до необхідності переходу від кустарних до індустріальних способів створення ПЗ і появи сукупності інженерних методів і засобів створення ПЗ, об'єднаних під загальною назвою "Програмна інженерія" (software engineering).

У процесі становлення та розвитку програмної інженерії можна виділити два етапи: 70-ті та 80-ті роки – систематизація та стандартизація процесів створення ПЗ (на основі структурного підходу), і 90-ті роки – початок переходу до складального, індустріального способу створення ПЗ (на основі

об'єктно-орієнтованого підходу).

Основна ідея програмної інженерії полягає в тому, що проектування ПЗ є формальним процесом, який можна вивчати та вдосконалювати. Освоєння та правильне застосування методів і засобів створення ПЗ дозволяють підвищити якість ІС, забезпечити керованість процесу проектування ІС та збільшити термін її експлуатації.

Завдяки унікальності та несхожості своїх складових частин, програмні системи принципово відрізняються від технічних систем, у яких переважають елементи, що повторюються. Комп'ютери самі по собі складніші, ніж більшість продуктів людської діяльності, оскільки кількість їх можливих станів дуже велика, що ускладнює розуміння, опис та тестування.

У програмних систем кількість можливих станів значно перевищує кількість станів комп'ютерів. Складність ПЗ є істотною, а не другорядною властивістю. Багато проблем розробки ПЗ походять з цієї складності та її нелінійного зростання при збільшенні розміру. Складність викликає труднощі у спілкуванні між розробниками, що веде до помилок у продукті, перевищення вартості розробки та затримки виконання графіків робіт. Вона також ускладнює розуміння усіх можливих станів програм, що знижує їх надійність, і стримує розвиток ПЗ та можливість додавання нових функцій.

Для успішної реалізації проекту об'єкт проектування (ПЗ) має бути адекватно описаний, тобто мають бути створені повні та несуперечливі моделі архітектури ПЗ, які визначають сукупність структурних елементів системи та зв'язків між ними, поведінку елементів системи під час їх взаємодії, а також ієрархію підсистем, що об'єднують структурні елементи.

Під моделлю розуміється повний опис системи ПЗ з певної точки зору. Моделі є засобами для візуалізації, опису, проектування та документування архітектури системи.

Моделі створюються для того, щоб зрозуміти та осмислити структуру і поведінку майбутньої системи, полегшити управління процесом її створення, зменшити можливий ризик, а також документувати прийняті проектні рішення.

Мова моделювання повинна включати: елементи моделі – фундаментальні концепції моделювання та їх семантику; нотацію – візуальне представлення елементів моделювання; керівництво по використанню – правила застосування елементів у рамках побудови тих чи інших типів моделей ПЗ.

Очевидно, що кінцева мета розробки ПЗ – це не моделювання, а отримання працюючих застосунків (коду). Діаграми, в кінцевому підсумку, є лише наочними зображеннями, тому, використовуючи графічні мови моделювання, дуже важливо розуміти, як вони допоможуть при написанні коду програм.

Наочність і строгість засобів структурного та об'єктно-орієнтованого аналізу дозволяють розробникам та майбутнім користувачам системи з самого початку неформально брати участь у її створенні, обговорювати та

закріплювати розуміння основних технічних рішень. Проте широке застосування цих методів та наслідування їх рекомендацій при розробці конкретних ІС стримувалося відсутністю адекватних інструментальних засобів, оскільки при неавтоматизованій (ручній) розробці всі їх переваги практично зводяться до нуля.

Дійсно, вручну дуже важко розробити та графічно представити суворі формальні специфікації системи, перевірити їх на повноту та несуперечність, а тим більше змінити. Якщо все ж таки вдається створити сувору систему проектних документів, її переробка при появі серйозних змін практично неможлива. Ручна розробка зазвичай породжувала такі проблеми: неадекватна специфікація вимог, нездатність виявляти помилки у проектних рішеннях, низька якість документації, що знижує експлуатаційні характеристики, затяжний цикл та незадовільні результати тестування.

Перераховані проблеми породили потребу в програмно-технологічних засобах спеціального класу – CASE-засобах, що реалізують CASE-технологію створення та супроводу ІС.

CASE-технологія є сукупністю методів проектування ІС та набором інструментальних засобів, що дозволяють в наочній формі моделювати предметну область, аналізувати цю модель на всіх стадіях розробки та супроводу ІС, і розробляти додатки відповідно до інформаційних потреб користувачів.

Мета навчального посібника – формування навичок самостійного практичного застосування сучасних методів та засобів об'єктно-орієнтованого аналізу та проектування інформаційних систем, заснованих на використанні візуального моделювання та CASE-засобів.

РОЗДІЛ 1. ОСНОВНІ ПОНЯТТЯ ТА ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1. Основні тенденції розвитку інформаційних технологій

Пройшли ті часи, коли програмісти були змушені економити комп'ютерний час та пам'ять на кожній програмній команді, функції чи модулі. Сьогодні спостерігається стрімке зростання потужності ЕОМ та складності завдань, які вирішуються. Акцент у розробці та реалізації програмного забезпечення переміщається з ефективного кодування та нестандартних рішень на простоту та швидкість розробки, а також на забезпечення взаєморозуміння всередині колективів розробників.

Інформація у сучасному світі перетворилася на один із найважливіших ресурсів, а інформаційні системи (ІС) стали необхідним інструментом практично в усіх сферах діяльності. Різноманітність задач, що вирішуються за допомогою ІС, призвела до появи безлічі різнотипних систем, що відрізняються принципами побудови та закладеними в них правилами обробки інформації.

Аналіз сучасного ринку ІС показує стійку тенденцію зростання попиту на інформаційні системи організаційного управління. Зокрема, попит продовжує зростати саме на інтегровані системи управління.

Перш ніж перейти до поняття інформаційних технологій, слід визначити термін "технологія". Спочатку це слово, яке виникло у стародавніх греків для позначення майстерності виготовлення речей, у сучасному трактуванні означає комплекс наукових та інженерних знань про способи та фактори виробництва для створення певного продукту або послуги. Засновником технології як окремої дисципліни вважається німецький вчений Йоганн Бекман (1739-1811), який написав ряд праць з технології та запровадив цей термін.

Інформаційна технологія – це системно-організована послідовність операцій (збір, реєстрація, передача, накопичення та обробка інформації), що виконуються над інформацією з використанням засобів і методів автоматизації. Головними процедурами в інформаційних технологіях є обробка інформації, тоді як інші процедури мають допоміжний характер.

Сучасні тенденції розвитку інформаційних технологій призводять до постійного зростання складності ІС, що створюються в різних галузях діяльності. Великі сучасні проекти ІС, як правило, мають такі особливості:

Складність опису (велика кількість функцій, процесів, елементів даних і складні взаємозв'язки між ними), що вимагає ретельного моделювання та аналізу даних і процесів;

Наявність сукупності тісно взаємодіючих компонентів (підсистем), що мають свої локальні завдання і цілі функціонування, використовують нерегламентовані запити до даних великого об'єму;

Відсутність прямих аналогів, що обмежує можливість використання типових проектних рішень і прикладних систем;

Необхідність інтеграції (композиції) існуючих та нових додатків;

Функціонування в неоднорідному середовищі на декількох апаратних платформах;

Роз’єднаність і різнорідність окремих груп розробників за рівнем кваліфікації та традиціями використання різних інструментальних засобів.

Для успішної реалізації ІС об’єкт проектування має бути адекватно описаний, що передбачає створення повних і несуперечливих функціональних і інформаційних моделей архітектури ІС. Це визначає сукупність структурних елементів системи та зв'язків між ними, поведінку елементів системи під час їх взаємодії, а також ієрархію підсистем, що об'єднують структурні елементи. Під моделлю розуміється повний опис системи ПЗ з певної точки зору.

Моделі є засобами для візуалізації, опису, проектування та документування архітектури системи. На думку одного з провідних фахівців в області об'єктно-орієнтованого підходу, Гради Буча, моделювання є центральною ланкою всієї діяльності зі створення якісного ПЗ. Моделі створюються для того, щоб зрозуміти та осмислити структуру і поведінку майбутньої системи, полегшити управління процесом її створення, зменшити можливий ризик, а також документувати прийняті проектні рішення.

Хороші моделі є основою взаємодії учасників проекту та гарантують коректність архітектури. Оскільки складність систем зростає, важливо мати у своєму розпорядженні ефективні методи моделювання. Хоча існує багато інших факторів, від яких залежить успіх проекту, наявність суворого стандарту мови моделювання є надзвичайно важливою.

Накопичений досвід проектування ІС показує, що це логічно складна, трудомістка і тривала за часом робота, що вимагає високої кваліфікації від фахівців. Усе це сприяє появі різних технологій програмування та програмно-технологічних засобів спеціального класу – CASE-засобів (Computer Aided Software Engineering), що реалізують CASE-технологію створення та супроводу ІС.

Термін CASE має дуже широке трактування. Первинне значення терміну CASE обмежувалося питаннями автоматизації розробки лише програмного забезпечення, але нині воно набуло нового сенсу і охоплює процес розробки складних ІС в цілому. Під терміном CASE-засоби розуміються програмні засоби, що підтримують процеси створення і супроводу ІС відповідно до інформаційних потреб користувачів, включаючи: аналіз і формулювання вимог, візуальне моделювання предметної області, аналіз цієї моделі на всіх етапах розробки і супроводу ІС.

У 70-80-х роках при розробці ПЗ широко застосовувалися структурні методи, що базуються на суворих формалізованих методах опису ПЗ та прийнятих технічних рішень. Нині такого ж поширення набувають об'єктно-орієнтовані методи. Ці методи ґрунтуються на використанні наочних графічних моделей: для опису архітектури ПЗ в різних аспектах (як статичної

структури, так і динаміки поведінки системи) використовуються схеми та діаграми. Наочність і суворість засобів структурного та об'єктно-орієнтованого аналізу дозволяють розробникам і майбутнім користувачам системи з самого початку неформально брати участь у її створенні, обговорювати та закріплювати розуміння основних технічних рішень.

1.2. Загальні поняття інформаційної системи

Під системою розуміють будь-який об’єкт, який одночасно розглядається як єдине ціле та як об’єднання різнорідних елементів, інтегрованих для досягнення визначених цілей. Серед численних визначень поняття системи (згідно з Вікіпедією) оберемо загальне. Тобто, система (грец. - «складене з частин», «з’єднання») – це сукупність взаємозалежних елементів, підпорядкованих єдиній меті, що забезпечує реалізацію функцій системи.

Інформаційна система – це взаємопов’язана сукупність засобів, методів і персоналу, що використовується для зберігання, обробки та видачі інформації з метою досягнення поставлених цілей.

Сучасне розуміння інформаційної системи передбачає використання персонального комп’ютера як основного технічного засобу для обробки інформації. У великих організаціях технічна база інформаційної системи може включати не лише персональні комп’ютери, але й мейнфрейми або суперЕОМ. Проте, технічне втілення інформаційної системи не має значення без врахування ролі людини, для якої призначена інформація і без якої її отримання та представлення є неможливим.

Очевидно, що існує різниця між комп’ютерами та інформаційними системами. Комп’ютери, оснащені спеціалізованим програмним забезпеченням, є технічною базою і інструментом для інформаційних систем. Обов’язковим компонентом будь-якої інформаційної системи є також персонал, який взаємодіє з комп’ютерами та телекомунікаціями.

1.3. Основні етапи розвитку інформаційних систем

З розвитком і вдосконаленням обчислювальної техніки, мов програмування та математичного забезпечення, автоматизовані системи обробки даних пройшли через кілька етапів еволюції.

Перші інформаційні системи виникли в 1950-х роках. Після війни комп'ютери використовувалися насамперед для оборонних завдань. Потужність комп'ютерів першого покоління була обмеженою, а програмування здійснювалося здебільшого на машинному коді. Основними завданнями були науково-технічні розрахунки, що потребували точного формулювання, а першими програмістами здебільшого були фізики та математики, які складали та вирішували рівняння. Вони ретельно розробляли алгоритми, готували детальну документацію, аналізували рішення колег та шукали математичні докази.

У ті ж роки вперше усвідомили важливість інформації як ресурсу для підприємств, організацій, регіонів і суспільства в цілому. Інформаційні системи використовувалися для обробки рахунків і розрахунку заробітної плати, реалізуючись на електромеханічних бухгалтерських машинах. Це призводило до скорочення витрат і часу на підготовку паперових документів.

1960-ті роки ознаменували зміну підходу до інформаційних систем. Проектування систем базувалося на інтуїтивних методах програмування, що спиралися на мистецтво, практичний досвід, експертні оцінки та дорогі експериментальні перевірки якості. Завдання часто змінювалися з урахуванням інформаційних потреб користувачів, що збільшувало час розробки, а документація оформлялася вже після того, як програма починала працювати.

Інформаційні системи почали використовувати для періодичної звітності за багатьма параметрами, що вимагало комп'ютерного обладнання широкого призначення, здатного виконувати численні функції. Мета використання – підвищення швидкості обробки документів, спрощення процедури обробки рахунків і розрахунку заробітної плати.

У 1970-х і на початку 1980-х років інформаційні системи та бази даних стали широко розповсюджуватися. Вартість зберігання одного біта інформації на комп'ютерних носіях стала меншою, ніж на традиційних носіях, що підвищило інтерес до комп'ютерних систем зберігання даних. Інформаційні системи почали використовувати як засіб управлінського контролю, що підтримує процес ухвалення рішень і прискорює його, особливо для вищої ланки управління.

З 1980-х до 2010-х років концепція використання інформаційних систем змінилася. Вони стали стратегічним джерелом інформації та використовувалися на всіх рівнях організації будь-якого профілю. Інформаційні системи, своєчасно надаючи необхідну інформацію, допомагали організаціям досягати успіху у своїй діяльності, створювати нові товари та послуги, знаходити нові ринки збуту, забезпечувати вигідне партнерство та організовувати випуск продукції за низькою ціною.

1980-ті роки характеризуються широким впровадженням персональних комп'ютерів у всі сфери людської діяльності, що створило великий і різноманітний контингент користувачів ПЗ. Це призвело до бурхливого розвитку користувацьких інтерфейсів та створення інформаційних систем нового покоління.

З'явилися мови програмування (наприклад, Ада), що враховували вимоги технології програмування. Почався активний процес стандартизації технологічних процесів, передусім документації, створюваної в цих процесах. На перший план вийшов об'єктно-орієнтований підхід до розробки ІС. Створювалися різні інструментальні середовища для розробки та супроводу ІС. Розвивалася концепція комп'ютерних мереж. Важливу роль почали відігравати засоби автоматизації розробки програм, що отримали назву CASE.

Підхід CASE надав розробникам ІС можливість використовувати

досконаліші інструменти, наприклад, мови четвертого покоління (4GL). Такі мови, як Visual Basic і Clarion, стали потужними та популярними 4GL-інструментами, які дозволяють збільшити продуктивність програмістів і зменшити кількість потенційних помилок.

1990-ті роки знаменувалися широким охопленням людського суспільства міжнародною комп'ютерною мережею. Персональні комп'ютери підключалися до неї як термінали, що поставило ряд проблем технологічного, юридичного та етичного характеру, пов'язаних із регулюванням доступу до інформації. Гостро постала проблема захисту комп'ютерної інформації та повідомлень, що передаються по мережі. Почався інтенсивний розвиток комп'ютерних технологій (CASE-технологій) розробки ІС другого покоління та пов'язаних з ними формальних методів специфікації програм.

Широке використання комп'ютерних мереж призвело до розвитку розподілених обчислень, дистанційного доступу до інформації та електронного обміну повідомленнями. Комп'ютерна техніка перетворилася на засіб інформаційного моделювання реального та уявного світу, здатний відповідати на запитання людей. Почався етап глибокої та повної інформатизації суспільства, що поставило перед технологією створення ІС нові складні завдання.

Незважаючи на високий потенціал CASE-технологій (підвищення продуктивності праці, покращення якості програмних продуктів, підтримка уніфікованого стилю роботи), далеко не всі розробники ІС, які використовують CASE-засоби, досягають очікуваних результатів. Головною причиною невдач є неадекватне розуміння суті програмування великих систем і неправильне застосування CASE-засобів.

1.4. Загальне порівняння інформаційних систем з традиційними програмними продуктами

Хоча інформаційні системи належать до категорії програмного забезпечення, вони відрізняються від стандартних прикладних програм і систем низкою важливих аспектів. В залежності від предметної області, інформаційні системи можуть значно відрізнятися за своїми функціями, архітектурою та реалізацією:

1. Інформаційні системи призначені для збору, зберігання та обробки інформації, тому їх основою є середовище зберігання та доступу до даних.

2. Інформаційні системи орієнтовані на кінцевого користувача, який може не мати високої кваліфікації в сфері використання обчислювальної техніки. Тому клієнтські додатки інформаційних систем повинні мати простий, зручний та функції, але водночас не дозволяє виконувати зайві дії.

Таким чином, розробка інформаційної системи включає вирішення двох основних завдань:

- створення бази даних, призначеної для зберігання інформації;
- створення графічного інтерфейсу користувача для клієнтських

додатків.

Розглянемо приклади деяких програмних засобів, які є або не є інформаційними системами:

1. 1С-Бухгалтерія 8.0: Використовується для формування бухгалтерської звітності підприємства перед податковими органами. Є інформаційною системою.
2. MS Excel: Універсальний програмний засіб для маніпуляцій з даними, представленими в табличній формі, автоматизації розрахунків, формування діаграм і аналізу даних. Не є інформаційною системою.
3. Книга MS Excel, що містить відомості про штатний розклад та працівників підприємства, оснащена макросами для розрахунку заробітної плати та формування платіжних відомостей. Є інформаційною системою.
4. Система комплексної автоматизації діяльності мережі роздрібних магазинів. Є інформаційною системою.
5. Реляційна СУБД DB-2 фірми IBM: Не є інформаційною системою.

1.5. Класифікація інформаційних систем

Інформаційні системи можна класифікувати за цілим рядом різних ознак. У основу цієї класифікації покладено найсуттєвіші характеристики, що визначають функціональні можливості та особливості побудови сучасних систем. Залежно від обсягу вирішуваних завдань, використовуваних технічних засобів і організації функціонування, інформаційні системи поділяються на кілька груп (класів). Розглянемо найбільш важливі з них:

За типом даних, що зберігаються, інформаційні системи поділяються на фактографічні та документальні. Фактографічні системи призначені для зберігання та обробки структурованих даних у вигляді чисел і текстів. У документальних системах інформація представлена у вигляді документів, що складаються з назв, описів, рефератів і текстів.

Ґрунтуючись на мірі автоматизації інформаційних процесів в системі управління фірмою, інформаційні системи поділяються на ручні, автоматичні та автоматизовані.

Ручні інформаційні системи характеризуються відсутністю сучасних технічних засобів обробки інформації, і всі операції виконуються людиною.

В автоматичних інформаційних системах всі операції з обробки інформації виконуються без участі людини.

Автоматизовані інформаційні системи передбачають участь у процесі обробки інформації як людини, так і технічних засобів, при цьому головну роль у виконанні рутинних операцій обробки даних відводиться комп'ютеру.

Залежно від характеру обробки даних, інформаційні системи поділяються на інформаційно-пошукові (довідкові) та інформаційно-вирішальні.

Інформаційно-пошукові системи призначені для задоволення інформаційних запитів користувачів. Інформація, знайдена відповідно до

запиту, не використовується безпосередньо в межах цієї ж системи, а видається користувачеві для його власних цілей. Пошук є однією з основних операцій у таких системах, тому їх також називають інформаційно-пошуковими системами (ІПС).

Інформаційно-вирішальні системи, крім того, здійснюють операції переробки інформації за певним алгоритмом. За характером використання вихідної інформації такі системи діляться на керуючі та порадницькі.

Керуючі інформаційні системи надають інформацію, яка безпосередньо використовується для прийняття рішень.

Порадницькі інформаційні системи надають інформацію, яка враховується при формуванні управлінських рішень, але не ініціює конкретні дії.

За сферою застосування інформаційні системи поділяються на наступні класи:

Інформаційні системи організаційного управління: призначені для автоматизації функцій управлінського персоналу як промислових підприємств, так і непромислових об'єктів (готелів, банків, магазинів). Основні функції таких систем включають оперативний контроль і регулювання, оперативний облік і аналіз, перспективне і оперативне планування, бухгалтерський облік.

Інформаційні системи управління технологічними процесами (ТП): служать для автоматизації функцій виробничого персоналу з контролю і управління виробничими операціями. В таких системах зазвичай передбачено наявність розвинених засобів вимірювання параметрів технологічних процесів (температури, тиску, хімічного складу тощо).

Інформаційні системи автоматизованого проектування (САПР): призначені для автоматизації функцій інженерів-проектувальників, конструкторів, архітекторів, дизайнерів при створенні нової техніки або технології. Основні функції таких систем включають інженерні розрахунки, створення графічної документації (креслень, схем, планів), створення проектної документації, моделювання проектованих об'єктів.

За масштабом інформаційні системи поділяються на однокористувацькі, групові та корпоративні.

Інтегровані (корпоративні) інформаційні системи: використовуються для автоматизації всіх функцій фірми та охоплюють весь цикл робіт від планування діяльності до збуту продукції. Вони включають ряд модулів (підсистем), які працюють в єдиному інформаційному просторі та виконують функції підтримки відповідних напрямів діяльності.

Існує класифікація інформаційних систем залежно від рівня управління, на якому система використовується.

Інформаційна система оперативного рівня: підтримує виконавців, обробляючи дані про угоди та події (рахунки, накладні, зарплата, кредити, потік сировини і матеріалів). Інформаційна система оперативного рівня є сполучною ланкою між фірмою та зовнішнім середовищем.

Інформаційні системи фахівців: підтримують роботу з даними і знаннями, підвищують продуктивність роботи інженерів і проектувальників. Завдання таких систем – інтеграція нових відомостей в організацію і допомога в обробці паперових документів.

Інформаційні системи рівня менеджменту: використовуються працівниками середньої управлінської ланки для моніторингу, контролю, ухвалення рішень і адміністрування. Основні функції цих інформаційних систем включають порівняння поточних показників з минулими, складання періодичних звітів за певний час, забезпечення доступу до архівної інформації тощо.

Стратегічна інформаційна система: комп'ютерна інформаційна система, що забезпечує підтримку ухвалення рішень щодо реалізації стратегічних перспективних цілей розвитку організації. Інформаційні системи стратегічного рівня допомагають вищій ланці управління вирішувати неструктуровані завдання та здійснювати довгострокове планування.

Архітектура "Файл-сервер" є історично першою архітектурою інформаційних систем. У цій архітектурі як виконувані модулі, так і дані розміщуються в окремих файлах операційної системи. Доступ до даних здійснюється за допомогою шляху (path) і використання файлових операцій (відкриття, зчитування, запис). Для зберігання даних використовується виділений сервер (окремий комп'ютер), який виконує роль файлового сервера. Виконувані модулі можуть зберігатися як на робочих станціях, так і на файловому сервері. У разі зберігання модулів на сервері спрощується їх адміністрування, але підвищуються вимоги до надійності мережі.

Архітектура "Клієнт-сервер" представляє не лише нову архітектуру, а й нову парадигму, що замінила застарілі концепції. Її сутність полягає в тому, що клієнт (виконуваний модуль) запитує різні сервіси згідно з визначеним протоколом обміну даними. На відміну від файлового сервера, клієнту не потрібно використовувати прямі шляхи операційної системи: він "не знає" їх, йому відоме лише ім'я джерела даних та інші специфічні відомості для авторизації на сервері. Сервер, який може фізично знаходитися на тому ж комп'ютері або на іншому кінці світу, обробляє запит клієнта і, здійснивши необхідні маніпуляції з даними, передає клієнту запитувану порцію даних.

У рамок архітектури "клієнт-сервер" існують два основні "діалекти": "тонкий" і "товстий" клієнт.

У систем на основі тонкого клієнта використовується потужний сервер баз даних – високопродуктивний комп'ютер із бібліотекою процедур, що дозволяють виконувати обчислення та реалізувати основну логіку обробки даних безпосередньо на сервері. Клієнтська програма має невисокі вимоги до апаратного забезпечення робочої станції. Основна перевага таких систем – відносна дешевизна клієнтських станцій.

Системи з товстим клієнтом, навпаки, реалізують основну логіку обробки на клієнті, тоді як сервер виконує роль чистого сервера баз даних, що забезпечує виконання стандартизованих запитів на маніпуляцію з даними

(читання, запис, модифікацію даних у таблицях реляційної бази даних). У таких системах вимоги до робочої станції вищі, а до сервера – нижчі. Перевага архітектури полягає в переносимості серверної компоненти на сервери різних виробників, оскільки всі промислові сервери баз даних реляційного типу підтримують роботу зі стандартизованою мовою маніпулювання даними SQL. Однак, кожен сервер має свою власну внутрішню вбудовану мову обробки даних, необхідну для реалізації логіки обробки.

1.6. Сфери застосування і приклади реалізації інформаційних систем

Раніше автоматизація бухгалтерського обліку була майже єдиною сферою застосування інформаційних систем, проте зараз спостерігається поширення інформаційних технологій у багатьох інших галузях. Розглянемо найбільш важливі завдання, які вирішуються за допомогою спеціалізованих програмних засобів.

Бухгалтерський облік. Це класична сфера застосування інформаційних технологій і найчастіше реалізоване завдання. Таке становище цілком зрозуміле. По-перше, помилка бухгалтера може бути дуже дорогою, тому очевидна вигода використання можливостей автоматизації бухгалтерії. По-друге, завдання бухгалтерського обліку досить легко формалізуються, що робить розробку систем автоматизації бухгалтерського обліку технічно нескладною проблемою.

Управління фінансовими потоками. Впровадження інформаційних технологій в управління фінансовими потоками обумовлено критичністю цієї сфери управління підприємством до помилок. Неправильне управління розрахунками з постачальниками та споживачами може спричинити кризу ліквідності, навіть за наявності добре налагодженої мережі закупівлі, збуту та ефективного маркетингу.

Управління складом, асортиментом, закупівлями. Автоматизація процесу аналізу руху товарів дозволяє відстежувати та фіксувати ті двадцять відсотків асортименту, які приносять вісімдесят відсотків прибутку.

Управління виробничим процесом. Основними механізмами є планування та оптимальне управління виробничим процесом. Автоматизовані рішення дозволяють грамотно планувати виробництво, враховувати витрати, здійснювати технічну підготовку та оперативно керувати процесом випуску продукції.

Управління маркетингом. Управління маркетингом передбачає збір та аналіз даних про фірми-конкуренти, їх продукцію та цінову політику, а також моделювання параметрів зовнішнього середовища для визначення оптимального рівня цін, прогнозування прибутку та планування рекламних кампаній.

Документообіг. Добре налагоджена система облікового документообігу відображає поточну виробничу діяльність на підприємстві і надає управлінцям можливість впливати на неї.

Оперативне управління підприємством. Інформаційна система для оперативного управління підприємством будується на основі бази даних, що містить всю доступну інформацію про підприємство. Така система є інструментом для управління бізнесом і зазвичай називається корпоративною ІС.

Надання інформації про фірму. Практично кожне підприємство, що себе поважає, має власний веб-сервер для вирішення двох основних завдань:

створення іміджу підприємства;

розвантаження довідкової служби компанії, надаючи необхідну інформацію про фірму, пропоновані товари, послуги і ціни.

Контрольні питання до розділу 1

1. Дайте визначення поняттю система.
2. Що таке інформаційна система?
3. Дайте визначення фактографічній системі.
4. Що таке інформаційно-пошукова (довідкова) система?

5.Що таке інтегрована (корпоративна) система?

6.Що таке моніторингова інформаціонна сістема?

7.Які ви знаєте сфери застосування ІС?

РОЗДІЛ 2. ТЕХНОЛОГІЇ ТА МЕТОДОЛОГІЇ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ

Зі збільшенням складності архітектури інформаційних систем зростає трудомісткість їх розробки, ускладнюються сценарії інтеграції та збільшуються витрати на супровід. Традиційне інтегроване середовище розробки (Integrated Development Environment, IDE) надає базові інструменти, такі як редактори коду, компілятори, засоби розробки призначеного для користувача інтерфейсу та відладчики. Однак ці інструменти не охоплюють ключові етапи життєвого циклу додатків, такі як визначення вимог, моделювання, тестування, розгортання та управління змінами.

Розробка інформаційних систем є циклічним процесом створення унікального продукту, який удосконалюється від версії до версії. Різні, не завжди послідовні етапи можуть взаємно впливати один на одного. По суті, створенню програмного продукту більше відповідає не модель управління проектом з чітким початком і кінцем, а модель управління всім життєвим циклом продукту.

Ітеративні методики розробки є більш придатними для циклічного процесу, оскільки вони передбачають постійну співпрацю між різними функціональними групами в команді проекту та кінцевими користувачами. Ці методики також забезпечують тісний взаємозв'язок між різними етапами життєвого циклу додатка, сприяючи більш ефективному управлінню розробкою та еволюцією програмного забезпечення.

2.1. Організація процесу розробки інформаційних систем

У цьому розділі визначаються фундаментальні концепції програмної інженерії (інженерії програмного забезпечення). Як і в будь-якій іншій інженерній дисципліні, основними складовими програмної інженерії є продукти (програмні системи) і процеси, які забезпечують їх створення.

Програмна інженерія - це система інженерних принципів, спрямованих на створення економічно ефективного програмного забезпечення, яке є надійним та продуктивним у реальних комп'ютерних середовищах. Це відносно молода дисципліна, її історія налічує трохи більше сорока років. У перші двадцять років (1970-1980-ті роки) в ній домінував класичний, процедурний підхід, тоді як з 1990-х по 2015-й роки став популярним об'єктно-орієнтований підхід до розробки програмного забезпечення.

У програмній інженерії розрізняють методи, засоби та процеси.

Методи забезпечують вирішення широкого спектру технічних завдань. Засоби (утиліти) програмної інженерії надають автоматизовану або автоматичну підтримку методів. Для спільного застосування утиліти можуть об'єднуватися в системи автоматизованої розробки ПЗ, які прийнято називати CASE-системами.

Процеси є "клеєм", що об'єднує методи та утиліти, забезпечуючи безперервний технологічний ланцюжок розробки.

Основні процеси життєвого циклу інформаційних систем (ІС) складаються з п'яти ключових етапів, що реалізуються під керівництвом основних учасників життєвого циклу програмного забезпечення. Основними учасниками є замовник, постачальник і розробник. До основних процесів належать:

- процес замовлення: визначає завдання та обов'язки замовника;
- процес постачання: охоплює завдання та обов'язки постачальника;
- процес розробки: описує завдання та обов'язки розробника;
- процес експлуатації: означає завдання оператора, відповідального за функціонування системи;
- процес супроводу: включає завдання організації, що надає послуги з підтримки та супроводу програмного продукту.

Розглянемо моделі процесів як сукупність основних елементів, таких як види діяльності, дії та завдання.

Діяльність – найбільший елемент, спрямований на досягнення значущої мети, який застосовується незалежно від прикладної області, розміру проекту або складності витрат. Діяльність складається з окремих дій.

Дія – проміжний елемент, що охоплює набір завдань, які утворюють етапний робочий продукт.

Завдання – найменший елемент, який зосереджений на чітко визначеній меті, наприклад, тестуванні модуля, що призводить до відчутного реального результату. Метою є створення програмного забезпечення у прийнятні строки і з необхідною якістю.

Підготовка. Зрозуміло, що будь-яка технічна робота вимагає належної підготовки, що полягає у тісній співпраці із замовником та іншими зацікавленими сторонами.

Важливим елементом є планування. План визначає послідовність інженерних робіт, описує технічні завдання, які потрібно виконати, можливі ризики, що можуть виникнути, необхідні ресурси, робочі продукти (моделі, документи, дані, звіти, форми тощо), які будуть створені, а також розклад роботи.

Наступним елементом є моделювання. У будь-якій діяльності моделювання є важливим інструментом для кращого розуміння загальної картини та створення ескізу майбутнього рішення. За потреби ескіз доповнюється деталями. Зазвичай моделювання включає дві дії: аналіз і проектування.

Конструювання - це діяльність об'єднує дві дії: генерацію програмного коду (вручну або автоматично) та тестування, яке необхідне для виявлення помилок у коді.

Розгортання. Програмне забезпечення (остаточна версія або частково завершений варіант) поставляється замовнику для оцінки отриманого продукту і надання зворотного зв'язку, що дає можливість покращити продукт.

2.2. Загальний життєвий цикл програмного забезпечення інформаційних систем

Розробка програмного забезпечення являє собою специфічну форму людської діяльності. Для ефективної її організації необхідно виділити окремі компоненти шляхом визначення набору завдань, які необхідно вирішити для досягнення кінцевої мети – створення якісної системи в межах встановлених термінів і ресурсів. Кожне завдання супроводжується допоміжною діяльністю, яку можна далі декомпозувати на менші складові. У результаті цього процесу має бути зрозуміло, як вирішувати кожну окрему підзадачу і все завдання в цілому, використовуючи наявні рішення для підзадач.

Прикладом діяльностей, необхідних для створення програмної системи, можуть бути: проектування тобто виділення окремих модулів і визначення зв'язків між ними з метою мінімізації залежностей між частинами проекту і досягнення кращої зрозумілості проекту в цілому; кодування, що полягає в розробці коду окремих модулів; розробка документації для користувачів.

Одним з основних понять методології проектування інформаційних систем є поняття життєвого циклу програмного забезпечення (ЖЦПЗ). ЖЦПЗ – це безперервний процес, який починається з моменту ухвалення рішення про необхідність створення системи і закінчується у момент її повного вилучення з експлуатації.

Протягом життєвого циклу створення інформаційних систем проходить через такі етапи, як аналіз предметної області, збір вимог, проектування, кодування, тестування, супровід та інші види діяльності.

При розробці інформаційних систем створюються та переробляються різні артефакти – інформаційні сутності, створені людиною, які виступають у ролі вхідних даних і виходять як результат різних діяльностей. Прикладами артефактів є: модель предметної області, опис вимог, технічне завдання, архітектура системи, проектна документація на систему і її компоненти, прототипи системи і компонентів, початковий код, документація для користувачів, документація адміністратора системи тощо.

Структура життєвого циклу програмного забезпечення за стандартом ISO/IEC 12207 включає три групи процесів:

- основні процеси життєвого циклу програмного забезпечення: придбання, постачання, розробка, експлуатація, супровід;
- допоміжні процеси: документування, управління конфігурацією, забезпечення якості, верифікація, атестація, оцінка, аудит, вирішення проблем;
- організаційні процеси: управління проектами, визначення, оцінка і покращення самого життєвого циклу, навчання.

Розробка включає всі роботи зі створення інформаційних систем і її компонентів відповідно до заданих вимог, включаючи оформлення проектної і експлуатаційної документації:

1.Планування і оцінка проекту. Під час цієї фази здійснюється виявлення

та детальне визначення властивостей, якими повинна володіти кінцева система, що дозволяє сформулювати остаточний концепт системи. У цей період приймаються рішення щодо розмірів, часу відгуку та інших критичних параметрів системи. Визначення характеристик системи здійснюється за допомогою двох основних категорій: вимог і специфікацій.

2.Аналіз системних і програмних вимог. Системний аналіз визначає роль кожного компонента в комп'ютерній системі та взаємодію цих компонентів між собою. Аналіз вимог стосується програмного елемента – програмного забезпечення. Під час цього процесу уточнюються та деталізуються його функціональні можливості, характеристики та інтерфейс.

3. Проектування алгоритмів, структур цих і програмних структур. Проектування полягає у створенні наступних представлень: архітектурної та модульної структури програмного забезпечення; алгоритмічної структури програмного забезпечення та організаціїструктури даних; вхідних та вихідних інтерфейсів, включаючи форми введення та виведення даних.

4.Реалізація (кодування) полягає у трансформації результатів проектування у програмний код, написаний мовою програмування.

5. Тестування відповідає полягає у виконанні програми з метою виявлення дефектів у функціональності, логіці та реалізації програмного продукту.

6. Введення в дію (супровід) внесення змін до існуючої інформаційної системи. Ці зміни мають на меті: виправлення помилок; адаптацію до змін зовнішнього середовища; удосконалення системи відповідно до вимог замовника.

7. Експлуатація включає роботи з впровадження компонентів інформаційної системи в експлуатацію, зокрема: створення баз даних і налаштування робочих місць користувачів; забезпечення конфігурації експлуатаційною документацією; проведення навчання персоналу; модифікацію та модернізацію інформаційної системи.

Вимоги - це характеристики, необхідні для вирішення проблеми або досягнення мети. Під час опису вимог використовуються такі поняття, як якість, можливості та інші характеристики системи, передбачуване її використання або середовище. Найкращою мовою для опису вимог, що забезпечує точне і всеосяжне вираження, є професійна мова експертів у сфері діяльності користувача.

Специфікація - це опис відомих зовнішніх характеристик поведінки системи мовою розробника.

Верифікація - це процес визначення того, чи відповідає поточний стан розробки вимогам на цьому етапі. Верифікація дозволяє оцінити відповідність параметрів розробки початковим вимогам. На кожному етапі слід переконатися, що зроблене відповідає плану і загальній логіці розробки - «Ми створюємо систему правильно».

Валідація (перевірка правильності) - це процес перевірки того, що реалізована система задовольняє пред'явленим вимогам і працює так, як

передбачалося - «Ми створюємо (створили) правильну систему».

2.3. Моделі життєвого циклу інформаційних систем

Модель життєвого циклу інформаційних систем (ЖЦІС) визначається специфікою програмного продукту та умовами, в яких він створюється і функціонує. Існують три основні стратегії конструювання інформаційних систем:

одноразовий прохід (водоспадна стратегія) – лінійна послідовність етапів розробки;

інкрементна (або ітеративна) стратегія – на початку процесу визначаються всі призначені для користувача і системні вимоги, а подальше конструювання здійснюється у вигляді послідовності версій, що включають заплановані покращення продукту;

еволюційна стратегія – система також розробляється у вигляді послідовності версій, проте на початковому етапі не всі вимоги визначені, вони уточнюються в процесі розробки нових версій.

Розглянемо коротко декілька моделей життєвого циклу.

Класична або каскадна модель

Рис. 2.1. Каскадна модель ЖЦІС

Етапи життєвого циклу інформаційних систем

Найбільш відомою і тривалий час широко застосовуваною моделлю життєвого циклу інформаційних систем була класична, або каскадна, модель (1970-1985 рр.), також відома як водоспадна модель (waterfall). Вперше вона була чітко сформульована у стандартах Міністерства оборони США (автор Вінстон Ройс, 1970). Ця модель передбачає послідовне виконання різних видів діяльності, починаючи з визначення вимог і закінчуючи супроводом, з чітким визначенням меж між етапами. На кожному етапі створюється набір документів, які передаються як вхідні дані для наступного етапу. Часто класичний життєвий цикл називають каскадною або водоспадною моделлю, підкреслюючи, що розробка розглядається як послідовність етапів, причому перехід на наступний етап відбувається тільки після повного завершення робіт

на поточному етапі.

Розглянемо більш детальніше етапи життєвого циклу інформаційних систем. Так, етап визначення стратегії завершується створенням документа, який чітко окреслює основні питання: що отримає замовник, якщо погодиться фінансувати проект; коли він отримає готовий продукт (графік виконання робіт); скільки це коштуватиме (графік фінансування на різних етапах робіт).

Результати цього етапу дозволяють відповісти на питання, чи варто продовжувати проект і які вимоги замовника можуть бути задоволені за певних умов. Іноді проект не має сенсу продовжувати через неможливість задовольнити певні вимоги. Якщо ухвалюється рішення про продовження проекту, етап аналізу розпочинається з уявлення про обсяг проекту і кошторис витрат.

Етап аналізу передбачає детальне дослідження бізнес-процесів (вимог і функцій, визначених на етапі вибору стратегії) та інформації, необхідної для їх виконання (сутностей, їх атрибутів і зв'язків). Вся інформація про систему, зібрана на етапі визначення стратегії, формалізується і уточнюється на етапі аналізу. Особливу увагу слід приділити повноті переданої інформації, виявленню суперечностей та усуненню невживаної або дубльованої інформації. Замовник зазвичай формує вимоги до окремих компонентів системи, тому необхідно забезпечити їх узгодженість. Фактично ці вимоги формують повне завдання на розробку.

Проектування включає дві основні дії: аналіз вимог і проектування. Результати цих дій зазвичай фіксуються на графічній мові моделювання. Передача інформації від аналітиків проектувальникам є інтерактивним процесом, під час якого відбувається уточнення інформації. При розробці схеми бази даних може змінитися інформаційна модель, отримана на етапі аналізу, наприклад, через нестабільність або низьку продуктивність проектного рішення.

Основна частина проектування бази даних полягає у побудові логічної і фізичної моделей даних. Інформаційна модель, створена на етапі аналізу, перетворюється спочатку в логічну, а потім у фізичну модель даних. Після цього створюється пробна база даних для розробників, які починають працювати з нею. Проектування бази даних тісно пов'язане з проектуванням модулів і додатків, оскільки бізнес-правила можуть створювати об'єкти в базі даних.

Головна мета проектування полягає у відображенні функцій, отриманих на етапі аналізу, в модулях інформаційної системи. Модулі визначаються в технічній специфікації програм, включаючи розмітку меню, вигляд вікон, гарячі клавіші та пов'язані з ними виклики.

Проектування процесу тестування зазвичай слідує за функціональним проектуванням і проектуванням схеми бази даних. На цьому етапі можуть використовуватися як складні, так і прості схеми тестування. Після завершення генерації модуля виконується автономний тест, що переслідує дві основні цілі: виявлення відмов модуля (жорстких збоїв); відповідність модуля

специфікації (наявність всіх необхідних функцій, відсутність зайвих функцій).

Після успішного проходження автономного тесту група згенерованих модулів проходить тести зв'язків для перевірки їх взаємного впливу. Далі група модулів тестується на надійність роботи через тести імітації відмов системи та напрацювання на відмову. Потім всі модулі проходять системний тест, що показує рівень якості продукту, включаючи тести функціональності та надійності системи.

Останній тест інформаційної системи - приймально-здавальні випробування, що передбачають показ системи замовникові через групу тестів, що моделюють реальні бізнес-процеси, щоб продемонструвати відповідність реалізації вимогам замовника.

На етапі розробки здійснюється тісна взаємодія проектувальників, розробників і груп тестерів. Тестер фактично стає членом групи розробки. Проектувальник виконує функції "ходячого довідника", постійно відповідаючи на питання розробників щодо технічної специфікації. На цьому етапі можуть змінюватися інтерфейси користувача і запити до даних. Взаємодія тестера і розробника без централізованої передачі частин проекту допустима, але тільки у випадку, якщо необхідно терміново перевірити якусь правку. Часто етапи розробки і тестування йдуть паралельно.

Під час розробки організовуються постійно оновлювані сховища готових модулів проекту і бібліотек, що використовуються при збірці модулів. Процес оновлення сховищ контролює одна людина. Одне сховище призначене для модулів, що пройшли функціональне тестування, інше - для модулів, що пройшли тестування зв'язків. Друге сховище використовується для зборки дистрибутиву системи і демонстрації його замовникові для контрольних випробувань або здачі етапів робіт.

Введення в дію (упровадження, розгортання) - це етап класичного життєвого циклу, спрямований на постачання розробленого продукту замовникові та супровід процесу його експлуатації. За статистичними даними, 65% супроводу пов’язано з удосконаленням ПЗ, 18% - з адаптацією, 17% - з виправленням помилок.

Експлуатація перекриває процес тестування, і система вводиться в експлуатацію поступово. Введення в експлуатацію проходить три фази: первинне завантаження інформації; накопичення інформації; вихід на проектну потужність.

Якщо вказати на позитивні сторони каскадного підходу, то можно назвати три особливості:

- забезпечення плану і погодинного графіку для всіх етапів проекту, впорядковуючи хід конструювання;
- на кожному етапі формується закінчений набір проектної документації, що відповідає критеріям повноти і узгодженості;
- логічна послідовність етапів робіт дозволяє планувати терміни завершення всіх робіт і відповідні витрати.

Серед недоліків каскадного підходу можно вказати:

- значне запізнення з отриманням результатів, оскільки реальні проекти часто вимагають відхилення від стандартної послідовності кроків;
- вимога повного завершення фази діяльності та закріплення результатів у вигляді детальної документації;
- узгодження результатів з користувачами здійснюється тільки після завершення кожного етапу робіт;
- користувачі і замовник не можуть ознайомитися з варіантами системи під час розробки і бачать результат тільки в самому кінці;
- вимоги до пз "заморожені" у вигляді технічного завдання на весь час розробки, що дозволяє користувачам вносити свої зауваження тільки після завершення робіт.

У більшості програмних проектів широко застосовується практика повторного використання деяких програмних модулів (компонентів). Це зазвичай трапляється у випадках, коли розробники проекту мають знання про раніше створені програмні продукти, які містять компоненти, що приблизно задовольняють вимогам нових компонентів, які розробляються.

Цей підхід базується на наявності великої бази існуючих програмних компонентів, які можна інтегрувати у нову систему. Часто такими компонентами є програмні продукти, що вільно продаються на ринку, і можуть бути використані для виконання певних спеціалізованих функцій.

Розглянемо більш детальніше основні етапи компонентного підходу. Так, по-перше, аналіз компонентів. Маючи специфіку вимог, на цьому етапі здійснюється пошук компонентів, які можуть задовольнити сформульовані вимоги. На стадії модифікації вимог аналізуються вимоги з урахуванням інформації про компоненти, отриманої на попередньому етапі. Вимоги коригуються для максимально ефективного використання можливостей відібраних компонентів.

Наступна стадія - проектування системи. Проектується структура системи або модифікується існуюча структура, використовуючи повторно застосовувані компоненти.

Розробка і складання системи - це етап безпосереднього створення системи. У рамках цього підходу складання системи є невід'ємною частиною розробки, а не окремим етапом.

Досить часто замовник не може чітко сформулювати детальні вимоги щодо введення, обробки або виведення даних для майбутнього програмного продукту. У таких випадках доцільно використовувати макетування (прототипування). Основна мета макетування полягає в знятті невизначеностей у вимогах замовника. Макетування - це процес створення моделі необхідного програмного продукту. Форми моделі можуть бути різноманітними, це і паперовий макет або макет на основі персонального комп’ютера, і працюючий макет (прототип) та існуюча програма (версія).

Як показано на рис. 2.2, макетування базується на багатократному повторенні ітерацій, у яких беруть участь замовник і розробник.

Рис 2.2 - Макетування (прототипування)

Ітеративна модель є класичним прикладом інкрементної стратегії розробки, яка об'єднує елементи послідовної водоспадної моделі з ітераційним макетуванням. У цій моделі кожна лінійна послідовність робіт створює інкремент (версію) програмного забезпечення, яке поставляється замовникові.

Але каскадний підхід страждає відсутністю гнучкості. Цей недолік усувається за допомогою каскадно-поворотного підходу, який дозволяє повернення до попередніх стадій і перегляд або уточнення раніше прийнятих рішень. Каскадно-поворотний підхід відображає ітеративний характер розробки ПЗ.

Ітеративні моделі передбачають розбиття системи на набір кроків, які розробляються за допомогою декількох послідовних проходів усіх робіт або їх частини. На першій ітерації розробляється незалежна частина системи. Велика частина або навіть повний цикл робіт проходиться для цієї частини, потім оцінюються результати, і на наступній ітерації або перша частина переробляється, або розробляється наступна частина, яка може залежати від першої. Також можливе поєднання доопрацювання першої ітерації з додаванням нових функцій. На кожній ітерації можна аналізувати проміжні результати робіт і реакцію на них усіх зацікавлених осіб, включаючи користувачів, і вносити коригування на наступних ітераціях. Кожна ітерація може містити повний набір видів діяльності від аналізу вимог до введення в експлуатацію чергової частини ПЗ. Замовникові не потрібно чекати повного завершення розробки системи, щоб отримати уявлення про неї. Компоненти, отримані на перших кроках розробки, задовольняють найбільш критичним вимогам і можуть бути оцінені на ранній стадії створення системи. Замовник може використовувати компоненти, отримані на перших кроках розробки, як прототипи для проведення експериментів та уточнення вимог до компонентів, які розроблятимуться пізніше.

Каскадна модель з можливістю повернення насамперед стає ітеративною, що дозволяє більш гнучко реагувати на зміни і потреби замовника, а також враховувати проміжні результати робіт для подальшого вдосконалення системи.

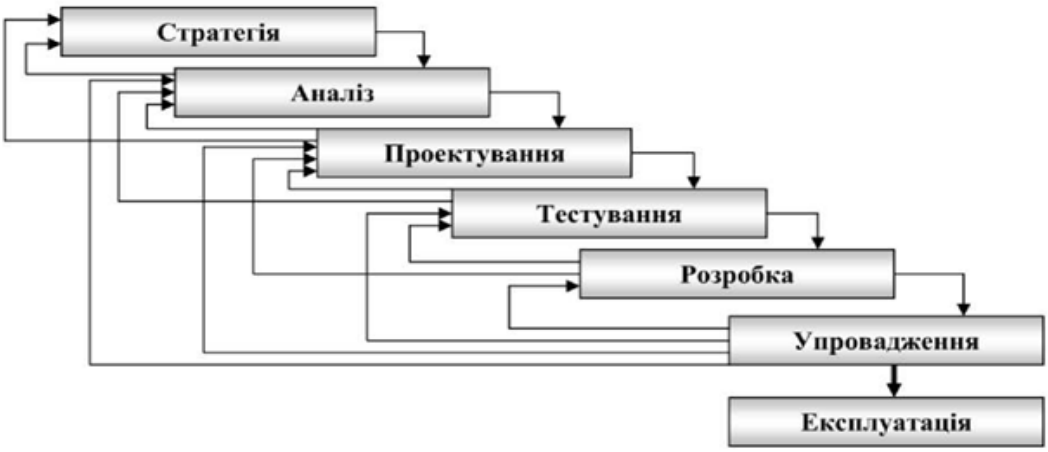

Рис. 2.3 - Можливий хід робіт по ітеративній каскадній моделі

Розвитком ітеративного підходу є спіральна модель життєвого циклу програмного забезпечення, яку запропонував Баррі Боем у 1988 році для зниження можливих ризиків розробки. Спіральна модель інтегрує поняття прототипування, що передбачає створення програми з частковою функціональністю кінцевого продукту. Створення прототипів здійснюється через декілька витків спіралі, кожен з яких включає фази "аналізу ризику", "деякого процесу" і "верифікації".

Спіральна модель (рис. 2.4) є класичним прикладом застосування еволюційної стратегії конструювання, створеної для подолання недоліків попередніх моделей. Кожен виток спіралі містить аналіз ризиків, реалізацію певного процесу і верифікацію результатів, що дозволяє поступово розширювати функціональність програмного продукту, мінімізуючи ризики і забезпечуючи гнучкість розробки.

Рис. 2.4 - Спіральна модель ЖЦІС

Виток спіралі відповідає створенню фрагмента або версії програмного забезпечення, на якому уточнюються цілі та характеристики проекту, визначається його якість і плануються роботи наступного витка спіралі. Таким чином, деталі проекту поступово поглиблюються і конкретизуються, що

дозволяє вибрати обґрунтований варіант для реалізації. Якщо розглядати позитивні риси цієї моделі, ми можемо вказати на те, що неповне завершення робіт на кожному етапі дозволяє перейти до наступного етапу без очікування повного завершення поточного. Окрім цього, кожен виток спіралі відповідає створенню фрагмента або версії ПЗ, де уточнюються цілі і характеристики проекту. Дозволяє явно враховувати ризик на кожному витку еволюції розробки. Використовує моделювання для зменшення ризику і вдосконалення програмного продукту.

Серед недоліків спірального циклу ми можемо визначити, підвищенні вимоги до замовника, визначення моменту переходу на наступний етап (для цього необхідно ввести тимчасові обмеження на кожний з етапів життєвого циклу), планування ґрунтується на статистичних даних, отриманих у попередніх проектах, та особистому досвіді розробників. Також у спіральній моделі відсутні фіксовані етапи, такі як розробка специфікації або проектування. Ця модель може включати будь-які інші моделі розробки систем на певному витку спіралі.

Контрольні питання до розділу 2

1. Дайте визначення життєвого циклу ПЗ.
2. Які ви знаєте основні процеси життєвого циклу ПЗ?
3. Що таке верифікація ПЗ?
4. Що таке валідація ПЗ?
5. Які ви знаєте основні стратегії конструювання ІС?
6. Перерахуйте основні переваги і недоліки каскадної моделі ЖЦІС.
7. Перерахуйте основні переваги і недоліки спіральної моделі ЖЦІС.

РОЗДІЛ 3. ТЕХНОЛОГІЇ СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ

У наступних розділі ми коротко розглянемо канонічне проектування інформаційних систем та три сучасні процеси розробки: уніфікований процес Rational (Rational Unified Process, RUP), екстремальне програмування (Extreme Programming, XP) та Scrum-технологію. Усі ці сучасні методики є прикладами ітеративних процесів, однак вони базуються на різних підходах до розробки програмного забезпечення.

RUP (Rational Unified Process) є прикладом так званого "важкого" процесу, який детально описаний і передбачає підтримку розробки початкового коду великою кількістю допоміжних дій. Це методологія, що акцентує увагу на документуванні та стандартизації процесів. Екстремальне програмування, навпаки, належить до "гнучких" (agile) методів розробки, які роблять акцент на майстерності розробників замість жорстко регламентованих процесів. Гнучкі методи уникають фіксації чітких схем дій, забезпечуючи велику гнучкість у кожному конкретному проекті.

Scrum-технологія, також належить до гнучких методів. Вона відома своєю простотою розуміння та великою гнучкістю. Хоча XP не була використана в повному обсязі навіть її авторами, деякі її практики, такі як парне програмування, колективне володіння кодом та рефакторинг коду, успішно впроваджуються на практиці. Тому ми розглянемо поліпшену технологію - Scrum.

Усі три методології є важливими інструментами сучасної розробки ПЗ, кожна з яких має свої переваги та недоліки, що робить їх придатними для різних типів проектів і команд.

3.1.Канонічне проектування інформаційних систем

Організація канонічного проектування інформаційних систем (ІС) базується на застосуванні каскадної моделі життєвого циклу інформаційних систем. Усі стадії та етапи робіт, які виконуються організаціями-учасниками, детально прописуються в договорах і технічних завданнях.

Більш детальніше розглянемо стадії та етапи створення інформаційних систем: формування вимог до інформаційних систем; визначення та узгодження вимог користувачів до системи; розробка концепції інформаційних систем; вивчення об'єкта автоматизації; розробка варіантів концепції інформаційних систем, що відповідають вимогам користувачів; оформлення звіту і затвердження концепції; технічне завдання; розробка та затвердження технічного завдання на створення інформаційних систем; ескізний проект; розробка попередніх проектних рішень для системи і її складових; створення ескізної документації на інформаційних систем; технічний проект; розробка детальних проектних рішень для системи і її частин; створення детальної документації на інформаційних систем та її складові; розробка робочої документації; підготовка робочої документації на інформаційні системи та її компоненти; введення в дію; комплектація

інформаційних систем необхідними програмними і технічними засобами; проведення попередніх випробувань; проведення дослідної експлуатації; проведення приймальних випробувань; супровід інформаційних систем; виконання робіт відповідно до гарантійних зобов'язань; післягарантійне обслуговування.

Саме ці стадії забезпечують структурований підхід до створення і впровадження інформаційних систем, гарантуючи, що всі аспекти проекту враховані, а система функціонує відповідно до вимог замовника.

Окремо слід розглянути обстеження об’єкту і обгрунтування створення інформаційних систем. Обстеження - це ретельне вивчення та діагностичний аналіз організаційної структури підприємства, його діяльності та існуючої системи обробки інформації. Отримані результати використовуються для обґрунтування розробки та поетапного впровадження систем; складання технічного завдання на розробку систем; розробки технічного та робочого проєктів систем.

На етапі обстеження доцільно виділити дві основні складові: визначення стратегії впровадження інформаційних систем та детальний аналіз діяльності організації.

Визначення стратегії

Головне завдання визначення стратегії - оцінка реального обсягу проекту, його цілей та завдань на основі виявлених функцій та інформаційних елементів об'єкта високого рівня, що підлягає автоматизації. Ці завдання можуть бути виконані як самим замовником інформаційних систем, так і з залученням консалтингових організацій. Цей етап передбачає тісну взаємодію з основними потенційними користувачами системи та бізнес-експертами.

Результатом першого етапу є документ (техніко-економічне обґрунтування проекту), в якому чітко визначено, що отримає замовник при фінансуванні проекту, строки отримання готового продукту (графік виконання робіт) та вартість проекту (для великих проектів має бути складений графік фінансування на різних етапах робіт).

На етапі детального аналізу діяльності організації вивчаються завдання, що забезпечують реалізацію функцій управління, організаційна структура, штати та зміст робіт з управління підприємством, а також характер підлеглості вищестоячим органам управління.

Одним із найскладніших завдань документообігу, яке водночас добре формалізується, є опис документообігу організації. Під час обстеження документообігу складається схема маршруту руху документів, яка повинна відображати: кількість документів; місце формування показників документу; взаємозв'язок документів при їх формуванні; маршрут і тривалість руху документу; місце використання і зберігання цього документу; внутрішні та зовнішні інформаційні зв'язки; висновки обстеження.

На основі результатів обстеження встановлюється перелік управлінських завдань, які доцільно автоматизувати та черговість їх розробки. Ці результати становлять об'єктивну основу для формування технічного

завдання на інформаційну систему.

Розробка технічного завдання передбачає вирішення таких завдань як встановлення загальної мети створення інформаційних систем, визначення складу підсистем та функціональних завдань. Технічне завдання - це основний документ, який визначає цілі, вимоги та початкові дані, необхідні для розробки автоматизованої системи управління.

Розробка та обґрунтування вимог до інформаційної бази, математичного і програмного забезпечення, комплексу технічних засобів (включаючи засоби зв'язку та передачі даних) включає встановлення загальних вимог до проектованої системи, переліку завдань створення системи та виконавців, визначення етапів створення системи та термінів їх виконанпня, роведення попереднього розрахунку витрат на створення системи та визначення рівня економічної ефективності її впровадження.

Розглянемо типові розділи та зміст технічного завдання. Так, Розділ 1. Загальні положення: повне найменування системи та її умовне позначення; шифр теми або номер договору; найменування підприємств-розробника і замовника; перелік документів, на підставі яких створюється інформаційна система, планові терміни початку і закінчення робіт, відомості про джерела і порядок фінансування робіт, порядок оформлення і пред'явлення замовникові результатів робіт зі створення системи, її частин та окремих засобів.

В Розділі 2. «Призначення і цілі створення (розвитку) системи». Вказується вид діяльності, що автоматизується, перелік об'єктів, на яких передбачається використання системи, найменування і необхідні значення технічних, виробничо-економічних та інших показників об'єкта, які мають бути досягнуті при впровадженні інформаційної системи.

В Розділі 3. «Характеристика об'єктів автоматизації» вказуються короткі відомості про об'єкт автоматизації, а також відомості про умови експлуатації та характеристики середовища функціонування інформаційної системи.

Розділ 4. «Вимоги до системи в цілому» присвячено встановленню вимог до структури і функціонування системи (перелік підсистем, рівні ієрархії, способи інформаційного обміну, взаємодія з суміжними системами, перспективи розвитку системи), вимог до персоналу (чисельність користувачів, кваліфікація, режим роботи, порядок підготовки), показників призначення (міра пристосовності системи до змін процесів управління та значень параметрів), вимог до надійності, безпеки, ергономіки, транспортабельності, експлуатації, захисту та збереження інформації, захисту від зовнішніх впливів, вимог до функцій перелік завдань, що підлягають автоматизації (часовий регламент реалізації кожної функції, вимоги до якості реалізації кожної функції) вимоги до видів забезпечення (математичне, інформаційне, лінгвістичне, програмне та організаційне забезпечення)

Розділ 5. «Склад і зміст робіт зі створення системи» містить інформацію про перелік стадій і етапів робіт, терміни їх виконання та склад організацій-виконавців робіт.

Розділ 6. «Порядок контролю і приймання системи» встановлює види,

склад, обсяг і методи випробувань системи.

В Розділі 7. «Вимоги до складу і змісту робіт з підготовки об'єкта до введення системи в дію» обов’язково здійснюється перетворення вхідної інформації у машиночитаний вигляд, вказуються зміни в об'єкті автоматизації, терміни і порядок комплектування та навчання персоналу.

До основних вимог документуванняє встановлення переліку документів, що підлягають розробці, а також перелік документів на машинних носіях. Документи та інформаційні матеріали, на підставі яких розробляється технічне завдання і система.

Ескізний проект передбачає розробку попередніх проектних рішень по системі та її частинам. Виконання стадії ескізного проектування не є обов'язковим. Якщо основні проектні рішення визначені раніше або очевидні для конкретної ІС та об'єкта автоматизації, ця стадія може бути виключена із загальної послідовності робіт.

Технічний проект системи - це технічна документація, що містить загальносистемні проектні рішення, алгоритми вирішення завдань, оцінку економічної ефективності автоматизованої системи управління та перелік заходів з підготовки об'єкта до впровадження. На цьому етапі здійснюється комплекс науково-дослідних і експериментальних робіт для вибору основних проектних рішень та розрахунку економічної ефективності системи.

На стадії «робоча документація» здійснюється створення програмного продукту та розробка всієї супроводжуючої документації. Документація повинна містити всі необхідні відомості для забезпечення введення інформаційної системи в дію, її експлуатації та підтримки експлуатаційних характеристик (якості) системи. Документація повинна бути належним чином оформлена, погоджена та затверджена.

До основних видів випробувань інформаційної системи загальноприйнято відносять:

- попередні випробування: визначення працездатності системи та вирішення питання про можливість її приймання в дослідну експлуатацію;
- дослідна експлуатація: визначення фактичних значень кількісних і якісних характеристик системи, готовності персоналу до роботи та коригування документації;
- приймальні випробування: визначення відповідності системи технічному завданню, оцінка якості дослідної експлуатації та вирішення питання про можливість приймання системи в постійну експлуатацію.

3.2. Загальний процес Rational

Уніфікований процес Rational (Rational Unified Process, RUP) представляє собою складну, детально розроблену ітеративну модель життєвого циклу розробки програмного забезпечення. Цей процес був розроблений компанією Rational Software, яка нині є частиною IBM, і є доповненням до мови моделювання UML.

Історично RUP є розвитком моделі процесу розробки, прийнятої в компанії Ericsson у 70-80-х роках XX століття. Ця модель була створена Іваром Джекобсоном, який у 1987 році заснував компанію Objectory AB для розвитку технологічного процесу розробки програмного забезпечення як окремого продукту, який можна було б використовувати в інших організаціях. Після злиття Objectory з Rational у 1995 році, розробки Джекобсона були інтегровані з роботами Волкера Ройса (Walker Royce), Філіппа Крухтена (Philippe Kruchten) та Греді Буча (Grady Booch), а також з універсальною мовою моделювання (Unified Modeling Language, UML).

До основних принципів RUP слід віднести:

- ітераційний і інкрементний підхід до створення програмного забезпечення;
- планування та управління проектом на основі функціональних вимог до системи - варіантів використання;
- побудова системи на базі архітектури програмного забезпечення.

Перший принцип є визначальним. Згідно з ним, розробка системи здійснюється у вигляді кількох короткострокових міні-проектів фіксованої тривалості (від 2 до 6 тижнів), які називаються ітераціями. Кожна ітерація включає власні етапи аналізу вимог, проектування, реалізації, тестування, інтеграції і завершується створенням працездатної системи.

Фази проектування в RUP в цілому аналогічні "водоспаду", але RUP виділяє в життєвому циклі чотири основні фази, в рамках кожної з яких можливе проведення кількох ітерацій.

Фаза початку проекту (Inception): Основна мета цієї фази - досягти компромісу між усіма зацікавленими сторонами щодо завдань проекту та ресурсів, які виділяються на його виконання. На цій фазі визначаються основні цілі проекту, керівник проекту, бюджет проекту, основні засоби виконання - технології, інструменти, ключові виконавці, а також апробація обраних технологій. На цю фазу може йти близько 10% часу і 5% трудомісткості одного циклу. Результатами цієї фази є:

- загальний опис системи: основні вимоги до проекту, його характеристики та обмеження;
- початковий бізнес-план;
- план проекту, що відображає стадії та ітерації;
- один або кілька прототипів.

Фаза проектування (Elaboration). Основна мета цієї фази - розробка стабільної базової архітектури продукту, яка дозволяє вирішувати поставлені завдання і використовується як основа для подальшої розробки системи. На цю фазу може йти близько 30% часу і 20% трудомісткості одного циклу.

Фаза побудови (Construction). Основна мета цієї фази - детальне уточнення вимог і розробка системи, що відповідає цим вимогам, на основі раніше спроектованої архітектури. У результаті повинна бути створена система, яка реалізує всі виділені варіанти використання. На цю фазу йде близько 50% часу і 65% трудомісткості одного циклу. Результатами цієї фази

є:

• модель варіантів використання (завершена принаймні на 80%), що визначає функціональні вимоги до системи;

• перелік додаткових вимог, включаючи нефункціональні вимоги;

• опис базової архітектури майбутньої системи, працюючий прототип;

• план розробки усього проекту, що відображає ітерації та критерії оцінки для кожної ітерації.

Фаза впровадження (Transition). Мета цієї фази - зробити систему повністю доступною кінцевим користувачам. На цій стадії відбувається розгортання системи у її робочому середовищі, бета-тестування, налаштування дрібних деталей під потреби користувачів. На цю фазу може йти близько 10% часу і 10% трудомісткості одного циклу.

Артефакти, що створюються під час проекту, можуть бути представлені у вигляді баз даних, таблиць з інформацією різного типу, різних видів документів, початкового коду та об'єктних модулів, а також моделей, що складаються з окремих елементів. Найбільш важливими з точки зору RUP артефактами є моделі, що описують різні аспекти майбутньої системи, більшість з яких є наборами діаграм UML.

До основних технік, що використовуються в RUP слід віднести:

• вироблення концепції проекту на його початку для чіткої постановки завдань;

• управління проектом за планом;

• зниження ризиків та відстеження їх наслідків, як можна раніше початок робіт з подолання ризиків;

• ретельне економічне обґрунтування усіх дій - робиться тільки те, що потрібно замовнику;

• як можна раніше формування базової архітектури;

• використання компонентної архітектури;

• прототипування, інкрементна розробка та тестування;

• регулярні оцінки поточного стану;

• управління змінами, постійна робота зі змінами, що надходять ззовні проекту;

• націленість на створення продукту, працездатного в реальному середовищі;

• орієнтація на якість;

• адаптація процесу під потреби проекту.

Як видно, фази в цілому відповідають моделі водоспаду, за винятком того, що фаза супроводу не розглядається як окрема. Загалом, розробка проекту RUP базується на чотирьох ключових ідеях:

• ітеративність процесу: кожна фаза, починаючи з розвитку, включає кілька ітерацій, на кожній з яких виконується частина аналізу, проектування, реалізації та тестування готового продукту або його частини;

• орієнтація на підсумкові цілі проекту (ітерації), виражені у вигляді варіантів використання (use cases) - сценаріїв взаємодії результатуючої

програмної системи з користувачами або іншими системами;

• на етапі розвитку приймаються ключові рішення щодо архітектури системи, її властивостей, функцій, використовуваних технологій тощо. в кінці цієї фази реалізується 10-30% системи, але всі основні рішення вже прийняті (до 70-80%);

• основою процесу розробки є плановані та керовані ітерації, обсяг яких визначається на основі архітектури (функціональність, що реалізується в рамках ітерації, та набір компонентів).

Уніфікований процес Rational (Rational Unified Process, RUP) являє собою складну, ретельно розроблену ітеративну модель життєвого циклу розробки програмного забезпечення. Цей процес є продуктом компанії Rational Software, нині частини IBM, і доповнює мову моделювання UML.

RUP є частиною інтегрованого комплексу інструментальних засобів Rational Suite, який включає:

• Rational Suite AnalystStudio: для визначення і управління вимогами;

• Rational Suite DevelopmentStudio: для проектування і реалізації ПЗ;

• Rational Suite TestStudio: для автоматичного тестування додатків;

• Rational Suite Enterprise: для підтримки повного життєвого циклу ПЗ;

До складу Rational Suite входять наступні компоненти:

• Rational Rose: засіб візуального моделювання (аналізу і проектування) з використанням UML;

• Rational XDE: засіб аналізу і проектування, інтегрований з MS Visual Studio .NET та IBM WebSphere Studio Application Developer;

• Rational Requisite Pro: засіб управління вимогами для організації спільної роботи групи розробників;

• Rational Rapid Developer: засіб швидкої розробки додатків на платформі Java 2 Enterprise Edition;

• Rational SoDA: засіб автоматичної генерації проектної документації;

• Rational Quantify: засіб кількісного визначення вузьких місць у програмі;

• Rational TestManager: засіб планування функціонального і навантажувального тестування;

• Rational TestFactory: засіб тестування надійності;

• Rational Quality Architect: засіб генерації коду для тестування.

Інструменти автоматичної генерації коду, використовуючи інформацію з діаграм класів і компонентів, формують файли описів класів. Створений таким чином алгоритм програми може бути уточнений шляхом програмування на відповідній мові (підтримувані мови: C++ і Java).

Хоча RUP є складною ітеративною моделлю розробки життєвого циклу ПЗ, вона може успішно застосовуватись, якщо на деяких етапах допускати певні "вільності".

Серед проблем кодування можна виділити небажання деяких розробників і замовників використовувати компоненти сторонніх виробників. Щоб уникнути повторного винаходу, при написанні коду можуть використовуватися готові програмні конструкції, типові рішення, шаблони

(design patterns).

Попри поліпшення, техніці RUP притаманні певні обмеження і недоліки:

- уніфікованість процесу робить його дещо заплутаним і неконкретним для різнорідних процесів, проектів і розробок;
- важкість для невеликих проектів і колективів розробників, особливо коли бюджет і терміни обмежені;
- вимога глибокого осмислення вимог, що може бути важким у реальних ситуаціях навіть на 70%;
- екстремальне програмування (XP-процес).

Останнім часом набирають популярність «гнучкі» методи розробки ПЗ (Agile Software Development), серед яких найпоширенішим є екстремальне програмування (extreme Programming, XP). Це полегшений (рухливий) процес, створений Кентом Беком у 1999 році. XP орієнтоване на малі та середні групи розробників, що працюють в умовах невизначених або швидко змінюваних вимог.

Основні принципи «живої» розробки ПЗ викладені в Маніфесті гнучкої розробки ПЗ, що з'явився у 2000 році: люди та їх взаємодія важливіші за процеси та інструменти; працююче ПЗ важливіше за вичерпну документацію; співпраця із замовником важливіша за обговорення контрактних умов; реакція на зміни важливіша за дотримання плану.

«Живі» методи виникли як протест проти надмірної бюрократизації розробки ПЗ, великої кількості побічних документів, які не є необхідними для отримання кінцевого результату, але оформляються для дотримання формальних пунктів контрактів на розробку.

Основна ідея XP - усунути високу вартість змін, характерну для об'єктно-орієнтованих додатків, використовуючи патерни (готові рішення) та реляційні бази даних. XP - це динамічний процес, що має справу зі змінами вимог на всьому протязі ітераційного циклу розробки, який складається з коротких ітерацій.

Кожна техніка XP важлива, і без її використання розробка не вважається XP. Основні практики XP:

- гра в планування (Planning game): швидке визначення зони дії наступної реалізації шляхом об'єднання ділових пріоритетів і технічних оцінок;
- часта зміна версій (Small releases): швидкий запуск у виробництво простої системи з поступовим додаванням нових функцій;
- метафора (Metaphor): розробка на основі простої історії про роботу системи;
- просте проектування (Simple design): проектування настільки просто, наскільки можливо на даний момент;
- тестування (Testing): безперервне написання тестів для модулів, які повинні виконуватися бездоганно;
- рефакторинг (Refactoring): наведення ладу в коді, видалення незрозумілих фрагментів, об'єднання класів;
- парне програмування (Pair programming): код пишеться двома

програмістами на одному комп’ютері;

• колективне володіння кодом (Collective ownership): будь-який розробник може покращувати будь-який код системи;

• безперервна інтеграція (Continuous integration): система інтегрується і будується багато разів на день;

• 40-годинний тиждень (40-hour week): не більше 40 годин роботи на тиждень;

• локальний замовник (On-site customer): представник замовника постійно присутній у команді розробників;

• стандарти кодування (Coding standards): правила, що забезпечують однакове представлення коду;

• відкритий робочий простір (Open workspace): розміщення команди в одному просторому приміщенні;

• зміна правил з потреби (Just rules): команда може змінювати правила, якщо всі члени згодні;

• недоліками XP деякі фахівці вважають нездійсненність у великомасштабних проектах, неможливість планування на довгу перспективу та передбачення результатів тривалих проектів у термінах якості і витрат.

Методологія Scrum

Однією з основних переваг методології XP є її гнучкість і простота для розуміння. Проте, навіть її творці рідко застосовували XP в повному обсязі. Замість цього успішно впроваджуються окремі практики XP, такі як парне програмування, колективне володіння кодом і рефакторинг. Ідея простого, ненадмірного дизайну проекту також мала значний вплив на розробку ПЗ. Більш практичним "гнучким" методом є Scrum.

У 1986 році японські фахівці Hirotaka Takeuchi і Ikujiro Nonaka опублікували статтю, де описали новий підхід до розробки продуктів і послуг, що не обов'язково стосуються програмного забезпечення. Основою підходу була злагоджена робота невеликої універсальної команди, яка розробляє проект на всіх етапах. Їх підхід порівнювався з грою в регбі, де вся команда рухається як єдине ціле, передаючи м'яч між гравцями як вперед, так і назад. На початку 90-х років цей підхід був адаптований до програмної індустрії та отримав назву Scrum (термін з регбі, що означає "сутичка"). У 1995 році Jeff Sutherland і Ken Schwaber представили опис цього підходу на конференції OOPSLA ‘95. Відтоді метод активно використовується в індустрії та широко описаний у літературі.

Метод Scrum дозволяє гнучко розробляти проекти невеликими командами (7 осіб ± 2) в умовах змінних вимог. Процес розробки є ітеративним і надає команді значну свободу. Спочатку формулюються вимоги до всього продукту, з яких обираються найактуальніші для наступної ітерації. Під час ітерації плани не змінюються, що забезпечує стабільність процесу. Ітерація триває від 2 до 4 тижнів і завершується створенням працездатної версії продукту, яку можна пред’явити замовнику для демонстрації та тестування. Після цього обговорюються результати і коригуються вимоги до продукту.

У методології Scrum визначено три основні ролі:

• власник продукту (Product Owner): менеджер проекту, який представляє інтереси замовника. Він відповідає за розробку початкових вимог (Product Backlog), своєчасну зміну вимог, призначення пріоритетів і дат постачання. Він не бере участі у виконанні ітерації;

• scrum-майстер (Scrum Master): забезпечує максимальну продуктивність команди, виконує Scrum-процеси, вирішує адміністративні завдання і обгороджує команду від зовнішніх впливів під час ітерації;

• scrum-команда (Scrum Team): група з 5-9 самостійних і ініціативних програмістів. Завдання команди - встановити досяжні цілі для ітерації на основі Product Backlog і виконати їх у відведений термін з належною якістю.

Методологія Scrum включає такі основні практики:

- Sprint Planning Meeting: проводиться на початку кожного спринту. Власник продукту, Scrum-майстер, команда та зацікавлені сторони визначають пріоритетні вимоги з Product Backlog для реалізації в рамках цього спринту. Формується Sprint Backlog. Далі Scrum-майстер і команда визначають, як досягти поставлених цілей, оцінюють трудомісткість завдань;
- Daily Scrum Meeting: щоденна п'ятнадцятихвилинна нарада для обговорення прогресу з часу попередньої зустрічі, корекції робочого плану і вирішення існуючих проблем. Кожен учасник команди відповідає на три питання: що я зробив з часу останньої зустрічі, які мої проблеми, що я планую зробити до наступної зустрічі;
- Scrum методологія дозволяє гнучко адаптувати процес розробки під змінні вимоги і забезпечує високу продуктивність роботи команди, зберігаючи при цьому чітку структуру і визначеність ролей та завдань.

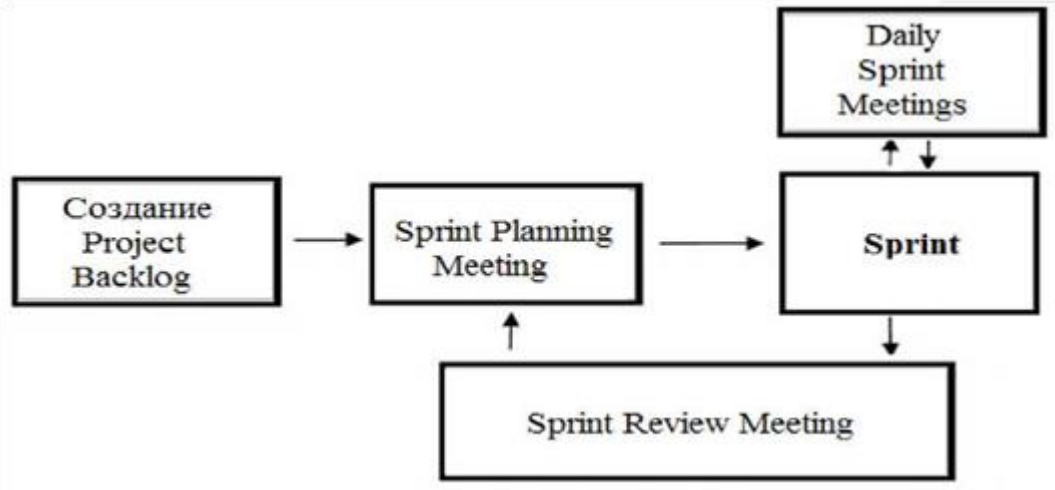

Рис. 3.1 - Ітераційна схема створення ПЗ

- Sprint Review Meeting. Проводиться наприкінці кожного спринту. Спочатку Scrum-команда демонструє Product Owner виконану роботу. Product Owner веде цю частину зустрічі і може запросити

всіх зацікавлених представників замовника. Він визначає, які вимоги з Sprint Backlog були виконані, і обговорює з командою і замовниками, як краще розставити пріоритети для наступної ітерації. У другій частині зустрічі проводиться аналіз минулого спринту під керівництвом Scrum-мастера. Scrum-команда аналізує позитивні та негативні аспекти спільної роботи, робить висновки і приймає рішення для поліпшення подальшої роботи. Також команда шукає способи підвищення ефективності. Після цього цикл повторюється.

Контрольні питання до розділу 3

1. Розкрийте сутність канонічної технології проектування інформаційних систем.
2. Перелічіть переваги та недоліки канонічної технології проектування ІС.
3. Перелічіть переваги та недоліки RUP-технології проектування ІС.
4. Перелічіть переваги та недоліки XP-технології.
5. Перелічіть переваги та недоліки SCRUM-технології.
6. В чому полягає ідея методології SCRUM та вкажіть її основні ролі.
7. Які стадії та етапи створення ІС із застосуванням канонічної технології ви знаєте?

РОЗДІЛ 4. УПРАВЛІННЯ ПРОЕКТОМ ПРИ ЗАГАЛЬНІЙ РОЗРОБЦІ ІНФОРМАЦІЙНИХ СИСТЕМ

4.1. Основні поняття управління проектом

Інформаційна система розробляється як певний проєкт. Багато аспектів управління проєктами та фази розробки проєктів є загальними та не залежать від предметної області або характеру проєкту (інженерний чи економічний).

Проєкт - це обмежена в часі цілеспрямована зміна окремої системи з чітко визначеними цілями, досягнення яких визначає завершення проєкту. Встановлюються вимоги до термінів, результатів, ризиків, витрат і ресурсів.

Метою будь-якого програмного проєкту є створення певного програмного продукту. Поняття "продукт" включає не лише текст на мові програмування і відкомпілюваний двійковий код, але й документацію, звіти про проміжні підсумки, результати перевірок, оцінки якості тощо. Ці елементи зазвичай називають артефактами.

Розглядаючи планування проєктів та управління ними, важливо розуміти, що йдеться про управління динамічним об'єктом. Тому система управління проєктом має бути гнучкою, щоб допускати можливість модифікацій без значних змін у робочій програмі.

При розробці великих проєктів керівникам доводиться відповідати на безліч питань, що починаються з "чому". Чому високий IQ недостатній для ефективного керування програмістами? Чому менше 20% проєктів з розробки ПЗ завершуються в строк і в межах бюджету, а третина проєктів анулюється до завершення?

Виявилося, що використання найкращих мов і технологій програмування, найсучасніших інструментів розробки та систем якості не гарантує успіху програмного проєкту. Людські якості забезпечують успіх проєкту, саме вони є чинником першорядної ваги. Головний елемент в розробці будь-якого проєкту - це люди, талановиті люди.

Таланти вирішують усе. Проста мобілізація засобів і зусиль вже не може забезпечити прогрес. Як казав Ф. Брукс, "Якщо проєкт не вкладається у терміни, додавання робочої сили затримає його ще більше". Ідею багатства тепер пов'язують не з грошима, а з людьми, не з фінансовим капіталом, а з "людським". Ринок праці перетворюється на ринок незалежних фахівців, які все більше знають про можливі варіанти вибору. Інтелектуальні працівники починають самостійно визначати свою ціну.

Суть усіх гнучких технологій (Scrum, XP тощо) полягає в тому, що вони декларують командну роботу, орієнтованість на людей та їх взаємодію, а не на процеси і засоби. Кожна розробка збирає навколо себе команду проєкту, що складається з зацікавлених осіб, які можуть грати наступні ролі:

- кінцеві користувачі. Здійснюють введення в систему, забезпечують зворотний зв'язок, проводять бета-тестування і допомагають визначити досягнення кінцевої мети;
- розробники**.** Один з розробників є керівником команди проєкту, її

лідером, який регулює потік інформації між членами команди і приймає більшість технічних рішень. Може бути окремий менеджер проєкту, відповідальний за адміністративні обов'язки;

• начальник відділу. Відповідає за достовірність даних, які надаються інформаційній системі;

• відповідальний за якість. Переконується, що проєкт досягає намічених цілей, відповідає опису системи, план тестування і план перетворення даних відповідають вимогам. За якість відповідає одна людина, але всі члени команди стежать за якістю;

• відповідальні за бета-тестування**.** Можливі кінцеві користувачі, які використовують спеціальні тести, розроблені відповідальним за якість.

Перед створенням основ технології керівникові проєкту необхідно вирішити всі організаційні проблеми:

• розділіть територіально розробників ПЗ і інших фахівців. Легкість комунікації високо цінується. Прекрасно, коли розробник може швидко уточнити деталі у замовника, але сусідство з менеджером по продажах може знизити продуктивність;

• забороніть, окрім менеджера проєкту, давати вказівки розробникам. Проект може вийти з-під контролю, якщо розробники виконуватимуть незаплановані розпорядження;

• визначить людину для обговорення завдань. Завдання по супроводу-розробці ПЗ виникають постійно, їх потрібно з'ясовувати негайно;

• надати для розробників окремий телефонний номер. Не залучайте розробників до секретарської роботи;

• не залучайте розробників до некваліфікованої праці. Розробник цінує свою кваліфікацію, і навряд чи легко знайдете нового розробника;

• не завантажуйте розробників одночасно декількома завданнями. Постійне відволікання знижує ефективність і викликає апатію;

• забезпечте для розробників навчання і обмін досвідом. Як керівник, ви не можете бути в курсі усіх останніх змін у світі розробки ПЗ. Ви можете вважати, що людині, яка вміє програмувати, більше не потрібно навчатися, оскільки досвід, здобутий в процесі роботи, є достатнім. Однак, якщо розробники не будуть оновлювати свої знання про новітні технології, ваше ПЗ швидко застаріє і стане перешкодою на шляху до нових досягнень на ринку.

Однією з поширених помилок є призначення керівника проекту, що не володіє відповідними технічними знаннями для реалізації цього проекту. Ця проблема зазвичай виникає на великих проектах, де потрібна значна команда програмістів. Часто існує технічний лідер, який може керувати проектом так само добре, як і вирішувати технічні питання.

У процесі аналізу вимог замовника важливо, щоб у переговорах брав участь один з членів команди розробників, бажано провідний технічний фахівець або технічний керівник проекту. Якщо у дискусіях бере участь лише адміністративна особа або керівник проекту, далекий від проблем безпосереднього кодування, виникає безліч питань, пов’язаних з оптимізацією

окремих операцій, створенням словника баз даних, системними вимогами до програмного забезпечення тощо. Участь технічних фахівців у обговоренні проекту може суттєво спростити проект завдяки приведенню окремих вимог користувача до вже існуючих і перевірених технологій.

Дотримання цих рекомендацій дозволить більш повно розглянути технічну сторону розробки, що допоможе уникнути багатьох помилок, пов'язаних з нерозумінням технічних особливостей проекту.

Основні обов'язки керівника проекту:

- сучасний керівник - це цілеспрямований організатор, який має натхнення і здатність створювати умови, що приваблюють таланти, а не просто службовців, що прагнуть зайняти робочі місця;
- людьми не потрібно "управляти". Завдання керівника - направляти людей, робити максимально продуктивними специфічні навички та знання кожного окремого працівника;
- відповідальність за щоденне керівництво проектом лежить на керівнику. Він може делегувати виконання певних завдань іншим, але не відповідальність за них;
- забезпечення єдності керівництва. Керівник не повинен дозволяти втручання ззовні, оскільки він несе особисту відповідальність за проект;
- розподіл ресурсів керівником базується на його відчутті та глибокому знанні завдання;
- переговори із замовником веде виключно керівник проекту. Він представляє інтереси команди і бере на себе зобов'язання від її імені;
- людський чинник є одним з найважливіших в управлінні проектами, але часто його ігнорують. Головне - це люди, талановиті люди. Керівник повинен уміти знаходити та виховувати талановитих розробників;
- інше завдання керівника - відводити різні проблеми від групи, щоб група розробників, працюючих над проектом, просто не знала про ті бурі, які бушують за стінами цього колективу, і не відволікалася на них. Неправильно поступають ті керівники, які, отримавши прочухан від замовника або власного керівництва, передають його тут же своїм виконавцям. Це абсолютно неприпустимо.

Фахівці в області оцінки і розвитку команд стверджують, що існує лише дев'ять командних ролей, баланс яких є вирішальним чинником успіху або невдачі в командній роботі. Приведемо деякі з них. Наприклад, генератор ідей. Оригінальний мислитель, який дає життя новим ідеям. Незалежний співробітник з розвиненою уявою, але подібно до інших людей має негативні риси вдачі - може бути надто чутливий до критики. Для успіху генератору ідей потрібні конструктивні стосунки з керівником або координатором групи.

Наступним буде дослідник ресурсів**.** Так само, як і генератор ідей, в змозі привнести нові ідеї до групи, але ці ідеї будуть запозичені ззовні, завдяки широким контактам. Дещо безцеремонний, гнучкий і шукає сприятливі можливості. До негативних якостей характеру відносяться лінь, самовдоволення.

Координатор**.** Зазвичай формальний лідер групи. Керує і направляє групу у бік досягнення цілей. Може заздалегідь визначити, хто з працівників хороший для виконання необхідних завдань. Зазвичай спокійний, упевнений і розпорядливий. Проте іноді схильний до зайвого домінування, і група стає продовженням його сильного «Я».

Мотиватор**.** Енергійний і в змозі впроваджувати ідеї. Бачить світ як проект, який вимагає впровадження. Зазвичай упевнений, динамічний, емоційний і імпульсивний. Мотор групи, але може бути дратівливим, невгамовним, нелюб'язним.

Аналітик**.** Оцінює пропозиції і займає позицію спостерігача за просуванням. Не дає групі рухатися неправильним шляхом. Обачний, безпристрасний, має аналітичний склад розуму. Може здаватися байдужим, незацікавленим, іноді стає надмірно критичним.

Натхненник команди**.** Прагне об'єднувати і вносити гармонію в стосунки між членами групи. Займає позицію того, що розуміє чужі проблеми, прагне допомогти і згладжує конфлікти. За вдачею людина добра, прагне налагоджувати неформальні стосунки. Проте буває нерішучим в складних або кризових ситуаціях.

Фахівець**.** Професіонал, самостійний прагне стати експертом у своїй області. Має високу професійну/технічну експертизу і знання, гордиться своєю роботою. Приносить вклад тільки у вузькій сфері своєї професійної експертизи.

Найважливішим неформальним макропоказником стану проекту є комунікації, їх якість і кількість. Тільки завдяки ефективним комунікаціям можна досягати синергетичного ефекту, який відрізняє команду від просто групи. Вже з XP-технології відомо, що освоєння нової технології парою програмістів, які здійснюють інтенсивний обмін знаннями, відбувається мінімум в 3 рази швидше, ніж у разі, коли ту ж роботу виконує один програміст.

Недостатня кількість комунікацій свідчить, як правило, про відсутність команди, кожен поглиблений у своє завдання і не цікавиться, що роблять його колеги. В результаті буде зроблено не те, що треба, а то, що буде зроблено, навряд чи вдасться інтегрувати в єдину систему.

Для побудови ефективної комунікації необхідно обов'язково враховувати індивідуальні особливості людей. Для кожного типу особи існує свій найбільш ефективний спосіб спілкування. Якщо той, що говорить не бере до уваги індивідуальні особливості того, що слухає, комунікації, як правило, заходять у безвихідь. При цьому буде відсутня (повністю або частково) передача інформації.

Не може бути ефективної команди, якщо учасники не знають і не уміють робити свою справу. Список чинників незрілості і неефективності співробітника:

1. Не дотримується інструкції.
2. Погано контролює час.

3. Не любить, коли контролюють його роботу.
4. Не звертає увагу на якість роботи.
5. Не може сконцентруватися на роботі.
6. Має особисті проблеми.
7. Перебільшує свої здібності.
8. Не виконує свою долю роботи.
9. Не любить змін в роботі.
10. Не повідомляє про помилки.
11. Не лояльний по відношенню до своєї компанії.

У більшості своїй ці негативні якості носять тимчасовий характер і походять від відсутності досвіду і недостатньої самостійності фахівця. Дружня підтримка і допомога, як правило, дозволяють впоратися з більшістю перерахованих проблем.

У свою чергу, ефективний програміст окрім технічних знань і умінь повинен володіти ще і особистими компетенціями, необхідними для командної роботи:

1. Займає активну позицію, прагне розширити свою відповідальність і збільшити особистий вклад в загальну справу
2. Постійно придбає нові професійні знання і досвід, висуває нові ідеї, спрямовані на підвищення ефективності досягнення спільних цілей, домагається поширення своїх знань, досвіду і ідей серед колег

3. Отримання задоволення від роботи

Задоволення від своєї діяльності, гордість за результати і прагнення поділитися цими почуттями з колегами є важливими для ефективної командної роботи.

4. Усвідомлення цілей.

Чітке розуміння особистих і загальних цілей, а також їх взаємозв'язку, сприяє наполегливому досягненню поставлених завдань.

5. Впевненість і об'єктивність

Упевненість у собі та своїх колегах, об'єктивна оцінка їх досягнень і уважне ставлення до інтересів та думок інших сприяють конструктивному вирішенню конфліктів.

6. Виклик як можливість

Кожна нова проблема розглядається як можливість підтвердити власний професіоналізм. Навіть сильні фахівці можуть завдати шкоди команді, якщо вони проявляють такі поведінкові патології:

- непорядність, брехливість, відсутність совісті та почуття справедливості;
- неповага до партнерів, схильність до негативних оцінок інших, грубість та відчуженість;
- завищена самооцінка та відчуття власної переваги;
- переоцінка свого внеску у загальну справу, що призводить до небажання працювати на рівні з колегами;
- взаємне розуміння між користувачем і розробником.

Ідеально, коли користувач має уявлення про технічну сторону завдання, а команда програмістів знайома з діяльністю користувача. Така комбінація знімає більшість питань з обговорення.

4.2. Планування програмного проекту ІС

Ефективне керівництво програмним проектом залежить від правильності планування робіт. План допомагає передбачати можливі проблеми і вжити захисних заходів для їх попередження і вирішення. Він створюється на початковому етапі проекту і розглядається менеджерами та інженерами-розробниками як керівний документ. План повинен детально описувати всі етапи роботи проекту.

Планування є багатокроковим ітераційним процесом. Важливо регулярно переглядати план, оскільки в проект безперервно надходять нові відомості. Враховуються контрактні зобов'язання фірми, вимоги замовника, бюджетні та тимчасові обмеження.

Планування починається з оцінки предметної області, розміру, трудовитрат і часу на розробку. Формується команда виконавців, розподіляються функції з урахуванням обмежень за часом, бюджетом, наявністю і можливостями співробітників, матеріально-технічним забезпеченням. Визначаються етапи розробки, контрольні віхи проекту і перелік артефактів.

Якщо виявляються внутрішні проблеми або змінюються вимоги замовника, можливий перегляд первинних оцінок проекту. Це може призвести до модифікації початкового графіку робіт. Якщо зміни впливають на терміни завершення або вартість проекту, узгоджуються нові обмеження з замовником. Після цього проект продовжується на наступному етапі.

План управління проектом повинен бути складений таким чином, щоб кожен знав свої завдання і терміни їх виконання. Існує безліч стандартів для таких планів. Більшість з них містять наступні розділи:

- вступ;
- огляд проекту: визначає проект без детального опису вимог;
- результуючі артефакти: список документів, початкових файлів і кінцевих продуктів;
- розвиток плану: напрями очікуваного розширення і змін;
- посилальні матеріали: джерела інформації;
- визначення і абревіатури;
- організація проекту;
- модель процесу: тип процесу розробки (водоспадний, спіральний, інкрементальний);
- організаційна структура: внутрішня організація команди;
- організаційні рамки і взаємозв'язки: способи взаємодії між організаціями;
- відповідальність за проект: межі відповідальності членів команди;
- аналіз ризиків;

• цілі і пріоритети: робоча філософія проекту;
• допущення, залежності і обмеження;
• управління ризиками;
• механізми моніторингу і контролю: хто і як буде контролювати проект;
• план розстановки кадрів;
• технічний процес;
• методи, інструменти і технології: обмеження на мови і використовувані інструменти;
• документація програмного забезпечення;
• функції супроводу проекту: дії для підтримки процесу розробки, такі як управління конфігурацією і забезпечення якості;
• розподіл робіт, графік і бюджет;
• розподіл робіт: як робота повинна розподілятися і надаватися після виконання;
• потреби в ресурсах: оцінка трудовитрат, апаратного і програмного забезпечення;
• виділення бюджету і ресурсів: розподіл ресурсів протягом життєвого циклу проекту;
• план-графік: розклад виконання різних етапів процесу;
• план повинен регулярно переглядатися у процесі виконання проекту.

Складання графіку робіт є однією з найважливіших функцій менеджера проекту. У цьому процесі менеджер визначає тривалість проекту, розподіляє необхідні ресурси та розпланувує послідовність завдань у часі. Зазвичай для цього використовуються спеціалізовані утиліти планування проектів.

Ці утиліти дозволяють:

- визначити критичний шлях (ланцюг завдань, що визначають тривалість всього проекту);
- встановити тривалість критичного шляху;
- визначити для кожного завдання найбільш ймовірну тимчасову оцінку;
- обчислити межі, що визначають тимчасове вікно для окремого завдання.

Першими етапами проекту є збір та аналіз вимог, які закладають фундамент для подальших завдань. Збір вимог проводиться з метою:

- визначення потреб замовника;
- оцінки здійсненності програмної системи;
- виконання економічного та технічного аналізу;
- визначення вартості та обмежень планування;
- створення системної специфікації.

Аналіз вимог дозволяє:

- уточнити функції та характеристики програмного продукту;
- визначити інтерфейс продукту з іншими системними елементами;
- встановити проектні обмеження програмного продукту;
- побудувати моделі процесу, даних, режимів функціонування

продукту;

- створити форми представлення інформації та функцій системи, які можна використати під час проектування.

Результати аналізу зводяться у специфікацію аналізу, що містить конкретизовані вимоги до програмного продукту.

Завдання по проектуванню та плануванню тестів можуть бути розподілені паралельно. Завдяки модульній природі ПЗ для кожного модуля можна передбачити окремий шлях для детального проектування, кодування та тестування. Після отримання всіх модулів ПЗ проводиться тестування інтеграції - об'єднання елементів в єдине ціле, а потім тестування правильності, яке забезпечує перевірку відповідності ПЗ вимогам замовника.

4.3. Управління персоналом

Найціннішим ресурсом програмного проекту є люди. Важливі не тільки технічні навички інженерів-розробників, але й уміння застосовувати ці навички в потрібний час і в потрібному місці. Це вимагає поєднання командної роботи та лідерства. Організація ефективної команди є складним завданням для менеджера проекту. Хороша команда повинна поєднувати професійні навички, досвід, згуртованість та дух товариства. Структура команди повинна стимулювати творчу роботу кожного її члена.

Керівник повинен організувати правильний підбір членів команди, враховуючи такі аспекти: досвід роботи у різних апаратно-програмних середовищах, знання мов програмування, освіта та професійний досвід, комунікабельність і здатність адаптуватися, Осооисті якості.

Освіта є комплексним показником початкових знань і навичок кандидата, а також його здібності до навчання. Досвід характеризує кінцеві знання і навички фахівця. Комунікабельність визначає можливості спілкування з колегами, керівниками та іншими зацікавленими особами. Здатність до адаптації може бути відображена у "послужному списку" - робочому стажі кандидата.

Оцінити особисті якості найскладніше. Це включає психологічний портрет, темперамент, ініціативність та цілеспрямованість. Вони визначають сумісність кандидата з колективом.

Особливо важливо правильно вибрати лідера команди. Він відповідає за технічне керівництво або адміністративне управління (іноді поєднуючи ці обов'язки). Лідер має бути в курсі повсякденної діяльності команди, забезпечуючи її ефективну роботу та співпрацю з керівництвом проекту. Він повинен ладнати з усіма членами колективу, згладжуючи напруженість і усуваючи неприємності.

При програмуванні без персоналізації всі робочі продукти (моделі, код, документація) вважаються власністю всієї команди, а не окремого співробітника.

Переваги розробки без персоналізації включають: спрощення процедур

перевірки та підвищення об'єктивності критики недоліків; заохочення невимушених обговорень робочих завдань, достоїнств і недоліків окремих рішень; підвищення дружніх стосунків та рівня щирості; швидке зростання майстерності завдяки роботі пліч-о-пліч; поліпшення якості та вдосконалення результатів роботи.

Для команди програмного проекту необхідна розвинена система взаємодії, яка включає спілкування та ефективні засоби зв'язку між співробітниками. Співробітники повинні інформувати один одного про хід роботи, прийняті рішення та зміни. Постійна взаємодія сприяє згуртованості і підвищенню якості роботи, оскільки співробітники обговорюють рішення і краще розуміють мотивацію колег.

На ефективність взаємодії впливають такі параметри:

1. Розмір/структура команди. Зі збільшенням числа учасників кількість зв'язків для взаємодії зростає квадратично. Наприклад, між трьома учасниками є три зв'язки, чотири учасники мають шість зв'язків, п'ять чоловік - десять зв'язків, тобто п учасників мають (п - 2) + ... + 1 = п(п - 1)/2 зв'язків (кожен з кожним). Отже, 50 чоловік повинні брати участь у 1225 взаємодіях, що практично неможливо.

На початку 1990-х років засновник фірми Borland, Філіп Кан, постулював формулу продуктивності розробки програмного забезпечення на конференції: "Продуктивність пропорційна кількості комунікацій в команді, але зі збільшенням числа учасників взаємодія стає складнішою, тому оптимальна кількість учасників команди має бути обмеженою.

COMDEX (Лас-Вегас, 1992 Він охарактеризував цю формулу продуктивності розробки програмного забезпечення як «Закон Філіпа». Відповідно до цього закону, продуктивність розробника ПЗ у команді з «N» осіб зменшується пропорційно до кубічного кореня з «N».

Графік (рис. 4.1) демонструє, що зі збільшенням кількості фахівців у команді розробників ПЗ продуктивність кожного з них знижується. Якщо дотримуватися закону Філіпа, то створення програмного коду обсягом 1 Гбайт за допомогою традиційних методів вимагало б величезних трудовитрат.

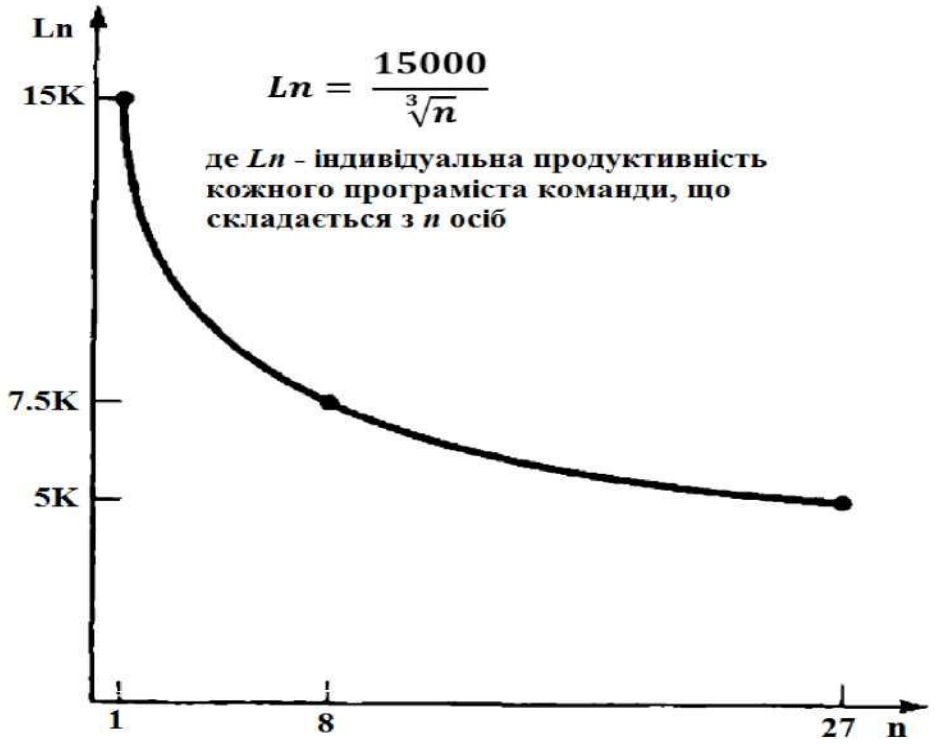

Рис. 4.1 - Закон Філіпа

З цього випливає, що традиційні методи розробки програмного забезпечення, які залежать від ручного кодування, поступово втрачають свою ефективність. Для створення складних і великих додатків майбутнього за допомогою сучасних засобів не вистачить ресурсів. Отже, техніка розробки ПЗ потребує кардинальних змін. Багато фахівців у цій галузі вважають, що об'єктно-орієнтовані методи частково вирішують цю проблему.

Для великих команд оптимальною альтернативою є поділ на менші групи, де кожна група відповідає за певну частину проекту і працює над нею автономно. Як правило, чисельність таких груп не перевищує восьми осіб. У таких компактних групах проблеми взаємодії мінімізуються. Для забезпечення координації з іншими групами у кожній з них виділяється один відповідальний співробітник. Така структура дозволяє зберігати переваги невеликих команд, забезпечуючи ефективну співпрацю великої кількості фахівців над створенням масштабних програмних продуктів.

У командах з горизонтальною організаційною структурою (де всі співробітники є рівноправними) взаємодія відбувається легше і ефективніше, ніж у командах з багаторівневою ієрархією, де є чіткий поділ на начальників і підлеглих. У багаторівневих командах комунікація проходить через рівні, у послідовності, що відповідає ієрархічній структурі.

Не існує єдиного універсального підходу для визначення оптимального складу групи розробників. Цей склад залежить від безлічі факторів: стилю управління в організації, специфіки предметної області, розміру проекту, професійних можливостей співробітників і багатьох інших чинників. Ось типові ролі:

Аналітик – відповідає за розробку та інтерпретацію вимог замовника;

повинен бути експертом у предметній області, працюючи в тісній співпраці з іншими членами команди.

Архітектор – відповідальний за проектування та розвиток архітектури продукту; один з найбільш кваліфікованих фахівців, який має досвід ухвалення стратегічних рішень; повинен мати навички програмування, оскільки його рішення реалізуються в коді.

Конструктор компонентів – основний творець компонентів, що формують продукт.

Фахівець з інтеграції – відповідає за збирання сумісних версій компонентів та перевірку їх спільної роботи, підтримуючи випуск версій продукту.

Фахівець з документації – документує всі реалізовані рішення, готуючи документацію для користувача.

Системний програміст – відповідає за створення та адаптацію програмних утиліт, що полегшують розробку в проекті.

Системний адміністратор – керує фізичними комп'ютерними ресурсами проекту.

Зрозуміло, не кожен проект вимагає виконання усіх цих ролей. У невеликих проектах співробітники можуть виконувати кілька ролей одночасно.

4.4. Процес розробки інформаційних систем

Найбільш активна дискусія щодо процесів розробки точиться навколо вибору між ітеративною та водоспадною моделями.

При організації роботи за водоспадною моделлю проект ділиться на етапи за типами робіт. Щоб створити програмне забезпечення, необхідно виконати певні дії: аналіз вимог, створення проекту, кодування і тестування. Такий річний проект може включати двомісячний етап аналізу, чотиримісячний етап проектування, тримісячний етап кодування і тримісячний етап тестування.

Ітеративний підхід ділить проект за функціональними характеристиками продукту. Можна взяти рік і розділити його на тримісячні ітерації. У першій ітерації реалізується чверть вимог, виконується повний цикл розробки: аналіз, проектування, кодування і тестування. До кінця першої ітерації система має чверть необхідної функціональності. Потім наступна ітерація додає ще функціональність і так далі.

При розробці за водоспадною моделлю після кожного етапу зазвичай відбувається формальна здача, але є можливість повернення назад. У процесі кодування можуть з'явитися обставини, що змушують повернутися до аналізу та проектування. Зазвичай, починаючи кодування, не слід вважати, що аналіз завершений. Рішення, прийняті на стадіях аналізу та проектування, можуть переглядатися пізніше, але такі зворотні потоки повинні бути зведені до мінімуму.

При ітеративному підході перед початком кожної ітерації проводиться

попереднє дослідження. Вимоги підлягають поверхневому аналізу, достатньому для розділення їх на ітерації. В результаті кожної ітерації має з'явитися інтегроване програмне забезпечення, готове до постачання, але може знадобитися стабілізаційний період для виправлення помилок.

Змішаний підхід (життєвий цикл поетапної доставки) передбачає виконання аналізу та проектування на верхньому рівні за водоспадною моделлю, а кодування і тестування – за ітеративною моделлю.

Більшість авторів публікацій щодо створення програмного забезпечення, особливо представників об'єктно-орієнтованого співтовариства, останніми роками схиляються до неприйняття водоспадної моделі. Основною причиною є те, що при використанні водоспадної моделі важко стверджувати, що проект рухається в правильному напрямку. Легко оголосити перемогу на ранньому етапі і приховати помилки планування. Ітеративний процес повторюється багаторазово, і якщо щось йде не так, ви своєчасно отримуєте сигнал про це.

З цієї причини рекомендується уникати водоспадної моделі у чистому вигляді і застосовувати поетапну доставку, якщо неможливо використати ітеративний метод у повному обсязі.

Особливо важливо, щоб при ітеративному процесі кожна ітерація завершувалася створенням протестованого, інтегрованого програмного продукту з максимально можливою якістю.

Загальноприйнятим підходом в ітеративній розробці є фіксація часу ітерацій. Якщо не вдається виконати все заплановане за час ітерації, слід вилучити частину функціональності, але не змінювати термін виконання. Це дозволяє вести розробку у постійному ритмі.

Ітеративна розробка передбачає переробку (рефакторинг) існуючого коду на останніх ітераціях проекту. У промисловому виробництві переробка вважається збитком, але в розробці програмного забезпечення часто вигідніше переробити код, ніж латати невдало спроектовану програму.

Ці технічні прийоми популяризовані в книзі «Extreme Programming», хоча вони застосовувалися і раніше, незалежно від використання XP або інших гнучких процесів.

4.5. Прогнозуюче і адаптивне планування інформаційних систем

Прогнозуючий підхід орієнтований на виконання значної частини робіт на початковому етапі проекту, щоб краще зрозуміти, що потрібно робити далі. Це дозволяє оцінити решту проекту з достатньою точністю. Процес прогнозуючого планування проекту розділяється на дві стадії. Перша стадія включає в себе складання планів, що важко передбачити, але друга стадія стає більш передбачуваною, оскільки плани вже готові.

Проте, існують активні дискусії щодо того, чи можна взагалі передбачити багато проектів. Суть цього питання полягає в аналізі вимог. Однією з найважливіших причин складності програмних проектів є труднощі в розумінні вимог до програмних систем.

Більшість програмних проектів стикається з істотним переглядом вимог на пізніх стадіях виконання. Це можна запобігти, заморозивши вимоги на ранньому етапі і не дозволяючи змін, але це призводить до ризику створення системи, яка більше не відповідає потребам користувачів.

Прибічники адаптивного підходу стверджують, що перегляд вимог неминучий. У багатьох проектах стабілізація вимог настільки складна, що прогнозуюче планування стає недоцільним. Це може бути наслідком того, що важко уявити можливості програмного продукту, або через непередбачувані зміни ринкових умов. Цей підхід підтримує адаптивне планування, де прогнозованість розглядається як ілюзія.

Відмінність між прогнозуючими і адаптивними проектами виявляється у способах обговорення стану проекту. Коли говорять, що проект виконується добре, оскільки йде за планом, мається на увазі прогнозуючий підхід. При адаптивній розробці не можна стверджувати "відповідно до плану", оскільки план постійно змінюється. Це не означає, що адаптивні проекти не плануються; планування займає значний час, але план розглядається як основна лінія проведення послідовних змін, а не як прогноз майбутнього.

Прогнозуючий підхід дозволяє укладати контракт з фіксованою функціональністю за фіксованою ціною. В такому контракті точно вказується, що має бути створено, скільки це коштує і коли продукт буде поставлений. В адаптивному плануванні така фіксація неможлива. Ви можете позначити бюджет і терміни постачання, але не можете точно зафіксувати функціональність продукту. Адаптивний контракт припускає співпрацю користувачів з командою розробників для регулярного перегляду функціональності та можливого припинення проекту при недостатньому прогресі.

Адаптивний підхід менш бажаний через те, що всі прагнуть до більшої передбачуваності. Проте передбачуваність залежить від точності, коректності і стабільності вимог. Звідси випливають дві важливі рекомендації:

1. Не складайте прогнозуючий план, поки не отримаєте точні, коректні і стабільні вимоги.

2. Якщо не можете отримати точні, коректні і стабільні вимоги, використовуйте адаптивне планування.

4.6. Вибір процесу розробки

Протягом останніх десяти років зріс інтерес до гнучких процесів розробки програмного забезпечення. Гнучкий (agile) – це широкий термін, що охоплює багато процесів, які мають загальні принципи, визначені Маніфестом гнучкої розробки програмного забезпечення (Manifesto of Agile Software Development). Прикладами таких процесів є XP (Extreme Programming), Scrum, FDD (Feature Driven Development) і DSDM (Dynamic Systems Development Method).

Гнучкі процеси відзначаються високою адаптивністю та орієнтацією на людину. Вони вважають, що найважливішим фактором успішного завершення

проекту є кваліфікація виконавців та їхня здатність ефективно співпрацювати. Значущість процесів та інструментів, що використовуються, стоїть на другому місці.

Гнучкі методи використовують короткі, обмежені за часом ітерації, зазвичай закінчуються через місяць або раніше. Вони не передбачають великого вкладення в документацію та не використовують UML в режимі проектування. UML частіше використовується в якості ескізів.

Гнучкі процеси не є надто формалізованими. Ваговиті процеси мають багато документації та постійний контроль під час виконання проекту. Гнучкий підхід припускає, що формалізм заважає проведенню змін і суперечить природі талановитих осіб.

Мартін Фаулер, прибічник ітеративного процесу розробки, зазначає, що ітеративний метод варто застосовувати лише в проектах, яким бажаєте успіху. Ітеративна розробка допомагає в ранньому виявленні можливих ризиків і покращенні керованості процесом розробки. Проте вона вимагає ретельного планування.

Уніфікований процес, розроблений компанією Rational (Rational Unified Process, RUP), часто згадується разом з UML. RUP є ітеративним процесом, несумісним з методологією водоспаду. Іноді його називають просто уніфікованим процесом (Unified Process, UP), коли організації застосовують термінологію і підхід RUP, але не використовують ліцензійні продукти Rational Software.

У Застосування RUP: Вибір шаблону розробкиПри впровадженні Rational Unified Process (RUP) першочергово необхідно визначити шаблон розробки (development case) – специфічний процес, який планується використати в рамках проекту. Шаблони розробки можуть варіюватися у значній мірі. Для вибору відповідного шаблону потрібен фахівець, добре знайомий з RUP, здатний адаптувати його під конкретні вимоги проекту. Як альтернатива, можна скористатися розподіленими за пакетами шаблонами, що постійно оновлюються і розширюються.

Ітеративна природа RUPНезалежно від обраного шаблону, RUP за своєю суттю є ітеративним процесом. Метод водоспаду не відповідає філософії RUP, хоча проекти, де використовуються водоспадні процеси, часто маскуються під RUP. RUP іноді називають просто уніфікованим процесом (Unified Process, UP), коли організації бажають застосовувати термінологію та загальний підхід RUP, не користуючись ліцензійними продуктами компанії Rational Software. RUP можна розглядати як продукт компанії Rational, заснований на UP, або вважати RUP і UP одним і тим самим.

Переваги ітеративної розробкиОднією з ключових переваг ітеративної розробки є можливість постійного вдосконалення процесу. Після завершення кожної ітерації слід проводити ретроспективний аналіз, збираючи команду для обговорення поточного стану справ і можливих покращень. Для коротких ітерацій достатньо двогодинних нарад. Після завершення проекту або його основної версії доцільно провести більш формальний ретроспективний аналіз,

який може тривати кілька днів. Детальніша інформація про проведення ретроспективних аналізів доступна на Retrospectives.com.

4.7. Налаштування UML під процес

При розгляді графічних мов моделювання, таких як UML, їх часто асоціюють з водоспадним процесом розробки. Водоспадний підхід зазвичай супроводжується детальною документацією, яка функціонує як прес-релізи між етапами аналізу, дизайну та кодування. Графічні моделі часто складають основну частину цих документів. Структурні методи 70-х і 80-х років також підкреслювали значення аналізу та дизайну моделей.

Незалежно від того, застосовуєте ви водоспадний метод чи ні, етапи аналізу, дизайну, кодування та тестування залишаються обов’язковими. Використання UML не обов’язково вимагає розробки документів або використання складних CASE-систем.

Аналіз вимог передбачає з'ясування очікувань клієнтів та користувачів від системи. UML пропонує кілька прийомів для цього етапу:

Прецеденти: Описують, як користувачі взаємодіють із системою.

Діаграми класів: Використовуються для створення концептуальної моделі та допомагають у розробці точного словника предметної області.

Діаграми діяльності: Показують робочий процес організації та взаємодію між програмним забезпеченням і користувачами.

Діаграми станів: Корисні для відображення життєвого циклу концепції з різними станами та подіями, що змінюють ці стани.

При аналізі станів важливо враховувати взаємодію з користувачами та клієнтами, які, як правило, не знайомі з UML. Головний ризик використання UML полягає у створенні діаграм, незрозумілих фахівцям предметної області, що може призвести до хибного почуття впевненості серед розробників.

На етапі проектування моделі використовуються більш активно і докладно. Деякі корисні прийоми включають:

- діаграми класів з точки зору ПЗ: відображають класи програми та їх взаємозв'язки;
- діаграми послідовності: використовуються для загальних сценаріїв і допомагають зрозуміти, що відбувається в програмі;
- діаграми пакетів: показують високорівневу організацію ПЗ;
- діаграми станів: для класів із складним життєвим циклом;
- діаграми розгортання: відображають фізичну конфігурацію ПЗ.

Моделювання зазвичай здійснюється на ранніх етапах ітерації та може виконуватися по частинах для різних функціональних розділів. Ітеративний процес означає, що моделі змінюються, а не створюються знову кожного разу.

4.8. Управління конфігурацією

Управління конфігурацією полягає в координації різних версій документації та програмного коду. Це включає: ідентифікацію змін; контроль змін; гарантію правильної реалізації змін; формування повідомлень про зміни.

Управління конфігурацією починається з початком проекту і закінчується з припиненням використання ПЗ. Програмний код зазвичай змінюється двома шляхами: додаються нові частини або оновлюються версії існуючих частин.

Інформацію, що виходить із процесу розробки програмного забезпечення, можна розділити на три категорії:

- комп’ютерні програми: виконувані коди;
- документи: описують програми як для технічного персоналу, так і для користувачів;
- структури даних: як зовнішні, так і внутрішні.

Мінімальна конфігурація ПЗ включає наступні базові елементи: системна специфікація, план проекту, специфікація вимог, попередній посібник користувача, специфікація проектування, лістинги початкових текстів, план і методика тестування, виконуваний код, опис бази даних, посібник користувача по налаштуванню, документи супроводу.

Управління конфігурацією забезпечує контроль змін впродовж усього життєвого циклу ПЗ, що є необхідним для успішного виконання проекту: посібник користувача щодо налаштування; документація для підтримки; звіти про програмні несправності; запити на підтримку; звіти про зміни, стандарти і методики розробки програмного забезпечення.

Управління конфігурацією передбачає застосування комплексних заходів для управління змінами протягом усього життєвого циклу програмного забезпечення. Зміни є невід'ємною частиною життєвого циклу програмного забезпечення. Замовники часто вимагають змін до вимог, а розробники прагнуть удосконалити технічні підходи.

Контрольні питання до розділу 4

1. Надати визначення терміну "об'єкт".
2. Перелічити ролі командного складу проєкту.
3. Перелічити основні обов'язки керівника проєкту.
4. Надати визначення термінам "адаптивні вимоги" та "адаптивний замовник".
5. Пояснити поняття ітеративного підходу до розробки програмного забезпечення.
6. Визначити відмінності між прогнозуючим та адаптивним плануванням.
7. Описати сутність уніфікованого процесу розробки програмного забезпечення (RUP).
8. Надати визначення терміну "управління конфігурацією".

РОЗДІЛ 5. ПОНЯТТЯ ТА АРХІТЕКТУРА СИСТЕМ КЕРУВАННЯ БАЗАМИ ДАНИХ

5.1. Основні поняття і визначення

Сучасні автори часто вживають терміни «банк даних» і «база даних» як синоніми, однак ці поняття розрізняються. Мають місце наступні визначення банку даних і бази даних:

Банк даних (БнД) – це система спеціальним образом організованих даних – баз даних, програмних, технічних, мовних, організаційно-методичних засобів, призначених для забезпечення централізованого нагромадження й колективного багатоцільового використання даних.

База даних (БД) – іменована сукупність даних, що відбиває стан об'єктів і їхнє відношення в розглянутій предметній області.

Термінологія в СКБД, та й самі терміни «база даних» і «банк даних», частково запозичені з фінансової діяльності. Це запозичення – не випадкове й пояснюється тим, що робота з інформацією і робота з грошовими масами багато в чому схожі, оскільки і там і там відсутня персоніфікація об'єкта обробки: дві банкноти номіналом у сто баксів настільки ж нерозрізнювані й взаємозамінні, як два однакових байти. Ви можете покласти гроші на деякий рахунок і надати можливість вашим родичам чи колегам використовувати їх для інших цілей; ви можете доручити банку оплачувати ваші витрати з вашого рахунка чи одержатигроші готівкою в іншому банку, і це будуть вже інші грошові купюри, але їхня цінність буде еквівалентна тій, котру ви мали, коли клали їх на рахунок.

Система керування базами даних (СКБД) – сукупність мовних і програмних засобів, призначених для створення, ведення й спільного використання баз даних багатьма користувачами.

Застосунки (застосування, додатки, програми) – програми, за допомогою яких користувачі працюють з базою даних, називаються.

Користувач бази даних – програма або людина, що звертається до бази даних мовою метаданих.

У загальному випадку з однією базою даних можуть працювати багато різних додатків. Наприклад, якщо база даних моделює деяке підприємство, то для роботи з нею може бути створений додаток, що обслуговує підсистему обліку кадрів, інший додаток може бути призначений для роботи підсистеми розрахунку заробітної плати співробітників, третій додаток працює як підсистема складського обліку, четвертий додаток займається плануванням виробничого процесу і т.ін. При розгляді додатків, що працюють з однією базою даних, передбачається, що вони можуть працювати паралельно і незалежно друг від друга, і саме СКБД призначена забезпечити роботу багатьох додатків з єдиною базою даних таким чином, щоб кожен із них виконувалося коректно,але враховуючи всі зміни в базі даних, внесені іншими додатками.

Запит – це процес звертання користувача до бази даних із метою введення, одержання або зміни інформації в базі даних.

Транзакція – послідовність операцій модифікації даних у базі даних, що переводить базу даних із одного несуперечливого стану в інший несуперечливий стан.

5.2. Архітектура та особливості роботи з системами керування базами даних

У процесі наукових досліджень, присвячених тому, як саме повинна бути скомпонована система керування базами даних, пропонувалися різні моделі. Найбільш життєздатною з них виявилася запропонована американським комітетом зі стандартизації ANSI (American National Standards Institute) трирівнева система організації БД.

Рівні організації системи керування базами даних за моделлю ANSI:

1. Рівень зовнішніх моделей – самий верхній рівень, де кожна модель має своє «бачення» даних. Цей рівень визначає точку зору на БД окремих додатків. Кожен додаток бачить і обробляє тільки ті дані, що необхідні саме цьому додатку. Наприклад, система розподілу робіт використовує зведення про кваліфікацію співробітника, але її не цікавлять зведення про оклад, домашню адресу і телефон співробітника; навпаки, саме ці зведення використовуються в підсистемі відділу кадрів.
2. Концептуальний рівень – центральна керуюча ланка, тут база даних представлена в найбільш загальному вигляді, що поєднує дані, використовувані всіма додатками, які працюють з даною БД. Фактично концептуальний рівень відбиває узагальнену модель предметної області (об'єктів реального світу), для якої створювалася база даних. Як будь-яка модель, концептуальна модель відбиває тільки істотні, з погляду обробки, особливості об'єктів реального світу (наприклад, вартість банкноти, а не колір).
3. Фізичний рівень – власне дані, розташовані в файлах чи в сторінкових структурах, розташованих на зовнішніх носіях інформації.

Архітектура ANSI дозволяє забезпечити логічну (між рівнями 1 і 2) і фізичну (між рівнями 2 і 3) незалежність при роботі з даними.

Логічна незалежність припускає можливість зміни одного додатка без корегування інших додатків, що працюють з цією ж базою даних.

Фізична незалежність припускає можливість переносу збереженої інформації з одних носіїв на інші при збереженні працездатності всіх додатків, що працюють з даною базою даних. Це саме те, чого не вистачало при використанні файлових систем.

А виділення концептуального рівня дозволило розробити апарат централізованого керування базою даних.

Процес проходження користувацького запиту

Як ми вже вказали, запит – це процес звертання користувача до бази даних із метою введення, одержання або зміни інформації в базі даних.

Розглянемо послідовність взаємодій:

1. Користувач посилає системі керування базами даних запит на одержання даних із БД {1}.
2. Аналіз прав користувача і зовнішньої моделі даних, що відповідає даному користувачу, й підтверджує чи забороняє доступ даного користувача до запитаних даних {2, 3}.
3. У випадку заборони на доступ до даних система керування базами даних повідомляє користувача про це {12} і припиняє подальший процес обробки даних, у противному випадку система керування базами даних визначає частину концептуальної моделі, що зачіпається запитомкористувача.
4. Система керування базами даних одержує інформацію про запитану частину концептуальної моделі {4, 5}.
5. Система керування базами даних запитує інформацію про місце розташування даних на фізичномурівні (файли чи фізичні адреси) {6}.
6. У системі керування базами даних повертається інформація про місце розташування даних у термінах операційної системи {7}.
7. Система керування базами даних просить операційну систему надати необхідні дані, використовуючи засоби операційної системи {8}.
8. Операційна система здійснює перекачування інформації з пристроїв зберігання і пересилає її в системний буфер {9}.
9. Операційна система оповіщає систему керування базами даних про закінчення пересилання {10}.
10. Система керування базами даних вибирає з доставленої інформації, що знаходиться в системномубуфері, тільки те, що потрібно користувачу, і пересилає ці дані в робочу областькористувача {11}.

БМД — це База Метаданих, саме тут і зберігається вся інформація про використовувані структури даних, логічної організації даних, права доступу користувачів і, нарешті, фізичне розташування даних. Для керування БМД існує спеціальне програмне забезпечення адміністрування баз даних, що призначено для коректного використання єдиного інформаційного простору багатьма користувачами. Чи завжди запит проходить повний цикл? Звичайно, ні. Система керування базами даних має досить розвинутий інтелект, що дозволяє їй не повторювати безглуздих дій.І тому, наприклад, якщо цей же користувач повторно звернеться до системи керування базами даних із новим запитом, то для нього вже не будуть перевірятися зовнішня модель і права доступу, а якщо подальший аналіз запиту покаже, що дані можуть знаходитися в системному буфері, то система керування базами даних здійснить тільки {11} і {12} кроки вобробці запиту.

Зрозуміло, механізм проходження запиту в реальних системах керування базами даних набагато складніший, але і ця спрощена схема показує, наскільки серйозними і складними повинні бути механізми обробки

запитів, підтримувані реальними системами керування базами даних.

Як будь-який програмно-організаційно-технічний комплекс, банк даних існує в часі й у просторі. Він має визначені стадії свого розвитку: проектування, реалізація, експлуатація, модернізація і розвиток, повна реорганізація.

На кожному етапі свого існування з банком даних зв'язані різні категорії користувачів. Визначимо основні категорії користувачів і їхню роль у функціонуванні банку даних.

Кінцеві користувачі. Це основна категорія користувачів, в інтересах яких і створюється банк даних. У залежності від особливостей створюваного банку даних коло його кінцевих користувачів може істотно розрізнятися. Це можуть бути випадкові користувачі, що звертаються до БД час від часу за одержанням деякої інформації, а можуть бути регулярні користувачі. Як випадкові користувачі можуть розглядатися, наприклад, можливі клієнти вашої фірми, що переглядають каталог вашої продукції чи послуг з узагальненим чи докладним описом того й іншого. Регулярними користувачами можуть бути ваші співробітники, що працюють зі спеціально розробленими для них програмами, що забезпечують автоматизацію їхньої діяльності при виконанні своїх посадових обов'язків. Наприклад, менеджер, що планує роботу сервісного відділу комп'ютерної фірми, має у своєму розпорядженні програму, що допомагає йому планувати і розподіляти поточні замовлення, контролювати хід їхнього виконання, замовляти на складі необхідні комплектуючі для нових замовлень.

Головний принцип полягає в тому, що від кінцевих користувачів не повинно вимагатися яких-небудь спеціальних знань в області обчислювальної техніки і мовних засобів.

Адміністратори банку даних. Це група користувачів, що:

- на початковій стадії розробки банку даних відповідають за його оптимальну організацію з погляду одночасної роботи багатьох кінцевих користувачів;
- на стадії експлуатації відповідають за коректність роботи даного банку інформації в багатокористувацькому режимі;
- на стадії розвитку і реорганізації відповідають за можливість коректної реорганізації банку без припинення його поточної експлуатації.

Розробники й адміністратори додатків. Це група користувачів, що функціонує під час проектування, створення й реорганізації банку даних. Адміністратори додатків координують роботу розробників при розробці конкретного додатка чи групи додатків, об’єднаних у функціональну підсистему. Розробники конкретних додатків працюють з тією частиною інформації з бази даних, яка потрібна для конкретного додатка.

Не в кожному банку даних можуть бути виділені всі типи користувачів. Наприклад, при розробці інформаційних систем з використанням настільних

Система керування базами даних (2 етап) адміністратор банку даних,

адміністратор додатків і розробник часто існували в одному обличчі. Однак при побудові сучасних складних корпоративних баз даних, що використовуються для автоматизації всіх чи більшої частини бізнесів-процесів у великій фірмі чи корпорації, можуть існувати і групи адміністраторів додатків, і відділи розробників. Найбільш складні обов'язки покладені на групу адміністратора БД.

5.4. Класифікація систем керування базами даних та їх функції

В науковій літературі та в технічній документації існує башато класифікацій систем керування базами даних, розглянемо деякі з них. Так, по-перше, це класифікація систем керування базами даних за видом програм.

Повнофункціональні системи керування базами даних (ПФСКБД) являють собою традиційні систем керування базами даних, що спочатку з'явилися для великих машин, потім для міні-машин і для ПЕОМ. З числа всіх систем керування базами даних сучасні ПФСКБД є найбільш численними і потужними по своїх можливостях. До ПФСКБД відносяться, наприклад, такі пакети, як Clarion Database Developer, DataBase, Dataplex, dBase IV, Microsoft Access, Microsoft FoxPro і Paradox R:BASE.

Звичайно ПФСКБД мають розвинутий інтерфейс, що дозволяє за допомогою команд меню виконувати основні дії з БД: створювати і модифікувати структури таблиць, вводити дані, формувати запити, розробляти звіти, виводити їх на друк і т.п. Для створення запитів і звітів не обов'язкове програмування, а зручно користатися мовою QBE (Query By Example - формулювання запитів за зразком). Більшість ПФСКБД включають також засоби програмування для професійних розроблювачів.

Деякі системи мають у якості допоміжних і додаткові засоби проектування схем БД чи CASE-підсистеми. Для забезпечення доступу до інших БД чи до даних SQL-серверів повнофункциональні системи керування базами даних мають факультативні модулі.

Сервери БД призначені для організації центрів обробки даних у мережах ЕОМ. Ця група БД у даний час менш численна, але їхня кількість поступова росте. Сервери БД реалізують функції керування базами даних, запитувані іншими (клієнтськими) програмами, звичайно за допомогою операторів SQL

Прикладами серверів БД є наступні програми: NetWare SQL (Novell), MS SQL Server (Microsoft), InterBase (Borland), SQLBase Server (Gupta), Intelligent Database (Ingress).

У ролі клієнтських програм для серверів БД у загальному випадку можуть використовуватися різні програми: ПФСКБД, електронні таблиці, текстові процесори, програми електронної пошти і т.д. При цьому елементи пари «клієнт-сервер» можуть належати одному чи різним виробникам програмного забезпечення.

У випадку, коли клієнтська і серверна частини виконані однією фірмою, природно очікувати, що розподіл функцій між ними виконаний раціонально. В інших випадках звичайно переслідується мета забезпечення доступу до

даних «за будь-яку ціну». Прикладом такого з'єднання є випадок, коли одна з повнофункціональних систем керування базами даних відіграє роль сервера, а друга система керування базами даних (іншого виробника) – роль клієнта. Так, для сервера БД SQL Server (Microsoft) у ролі клієнтських (фронтальних) програм можуть виступати більшість систем керування базами даних, такі як dBASE IV, Biyth Software, Paradox, DataEase, Focus, 1-2-3, MDBS III, Revelation іінші.

Засоби розробки програм роботи з БД можуть використовуватися для створення наступних програм: клієнтських програм; серверів БД і їхніх окремих компонентів; користувацьких додатків.

Програми першого і другого виду досить нечисленні, тому що призначені,головним чином, для системних програмістів. Пакетів третього виду набагато більше, але все ж менше, ніж повнофункціональних систем керування базами даних.

До засобів розробки користувацьких додатків відносяться системи програмування, наприклад Clipper, різноманітні бібліотеки програм для різних мов програмування, а також пакети автоматизації розробок (у тому числі системтипу клієнт-сервер). У числі найбільш розповсюджених можна назвати наступніінструментальні системи:

Delphi і Power Builder (Borland), Visual Basic (Microsoft), SILVERRUN (Computer Advisers Inc.), S-Dcsignor (SDP і Powersoft) і ERwin (LogicWorks).

Крім перерахованих засобів, для керування даними й організації обслуговування БД використовуються різні додаткові засоби, наприклад, монітори транзакцій

Друга класифікація систем керування базами даних Класифікація СКБД за моделлю даних. По використовуваній моделі даних системи керування базами даних (як і БД), розділяють на ієрархічні, мережеві, реляційні, об’єктно-орієнтовані й інші типи. Деякі системи керування базами даних можуть одночасно підтримувати кілька моделей даних.

З погляду користувача, система керування базами даних реалізує функції збереження, зміни (поповнення, редагування і видалення) і обробки інформації, а також розробкий одержання різних вихідних документів. Для роботи з інформацією, що зберігається в базі даних, система керування базами даних надає програмам і користувачам наступні два типи мов:

- мова опису даних - високорівнева непроцедурна мова декларативного типу, призначена для опису логічної структури даних;
- мова маніпулювання даними - сукупність конструкції, що забезпечують виконання основних операцій по роботі з даними: введення, модифікація, вибірка даних по запитах.

Названі мови в різних системах керування базами даних можуть мати відмінності. Найбільше поширення одержали дві стандартизовані мови: QBE (Query By Example) - мова запитів за зразком і SQL (Structured Query Language) - структурована мова запитів.

QBE в основному має властивості мови маніпулювання даними, SQL

поєднує у собі властивості мов обох типів - опису і маніпулювання даними.

Перераховані вище функції систем керування базами даних, у свою чергу, використовують наступніосновні функції більш низького рівня, що назвемо низькорівневими:

- керування даними в зовнішній пам'яті;
- керування буферами оперативної пам'яті;
- керування транзакціями;
- ведення журналу змін у БД;
- забезпечення цілісності і безпеки БД.

Наведемо коротку характеристику необхідності й особливостям реалізації перерахованих функцій у сучасних системах керування базами даних.

Керування даними в зовнішній пам'яті. Реалізація функції керування даними в зовнішній пам'яті в різних системах може розрізнятися:

- на рівні керування ресурсами: використовуючи файлові системи ОС чи безпосереднє керування пристроями ПЕОМ;
- по логіці самих алгоритмів керування даними: В основному методи й алгоритми керування даними є «внутрішньою справою» систем керування базами даних і прямого відношення до користувача не мають. Якість реалізації цієї функції найбільш сильно впливає на ефективність роботи специфічних інформаційних систем, наприклад, з величезними БД, зі складними запитами, великим обсягом обробки даних.

Керування буферами оперативної пам'яті. Необхідність буферизації даних і як наслідок реалізації функції керування буферами оперативної пам'ятіобумовлене тим, що обсяг оперативної пам'яті менше обсягу зовнішньої пам'яті.Буфери являють собою області оперативної пам'яті, призначені для прискорення обміну між зовнішньою й оперативною пам'яттю. У буферах тимчасово зберігаються фрагменти БД, дані з який передбачається використовувати при звертанні до систем керування базами даних чи планується записати в базу після обробки.

Керування транзакціями. Механізм транзакцій використовується в системах керування базами даних для підтримки цілісності даних у базі.

Транзакція – послідовність операцій модифікації даних у БД, що переводить БД із одного несуперечливого стану в інший несуперечливий стан. Тобто транзакцією називається деяка неподільна послідовність операцій над даними БД, що відслідковується системами керування базами даних від початку і до завершення. Якщо за певними причинами (збої і відмови устаткування, помилки в програмному забезпеченні, включаючи додатки) транзакція залишається незавершеною, то вона скасовується.

Транзакція має три основних властивості:

- атомарність (виконуються всі включені в транзакцію операції або одна);
- серіалізованість (відсутній взаємний вплив виконуваних у той самий час транзакцій);

- довговічність (навіть крах системи не призводить до втрати результатів зафіксованої транзакції).

Прикладом транзакції є операція переведення грошей з одного рахунка на іншій у банківській системі. Тут необхідний, принаймні, двокроковий процес. Спочатку знімають гроші з одного рахунку, потім додають їх до іншого рахунку.Якщо хоча б одна з дій не виконається успішно, результат операції виявиться невірним і буде порушений баланс між рахунками.

Контроль транзакцій важливий в однокористувацьких і в багатокористувацьких систем керування базами даних, де транзакції можуть бути запущені паралельно. В останньому випадку говорять про серіалізованість транзакцій. Під серіалізацією паралельно виконуваних транзакцій розуміється складання такого плану їхньоговиконання (серіального плану), при якому сумарний ефект реалізації транзакційеквівалентний ефекту їхнього послідовного виконання.

При паралельному виконанні набору транзакцій можливе виникнення конфліктів (блокувань), розв'язання яких є функцією систем керування базами даних. При виявленні таких випадків звичайно виконується «відкат» шляхом скасування змін, зроблених однією чи декількома транзакціями.

Ведення журналу змін у БД (журналізація змін) виконується систем керування базами даних для забезпечення надійності збереження даних у базі при наявності апаратних збоїві відмовлень, а також помилок у програмному забезпеченні.

Журнал систем керування базами даних - це особлива БД чи частина основної БД, безпосередньо недоступна користувачу і використовувана для запису інформації про всі зміни бази даних. У різних системах керування базами даних у журнал можуть заноситися записи, що відповідають змінам у системі керування базами даних на різних рівнях: від мінімальної внутрішньої операції модифікації сторінки зовнішньої пам'яті до логічної операції модифікації БД (наприклад, вставки запису, видалення стовпця, зміни значення в полі) – і навіть транзакції.

Для ефективної реалізації функції ведення журналу змін у БД необхідно забезпечити підвищену надійність збереження і підтримки в робочому стані самого журналу. Іноді для цього в системі зберігають кілька копій журналу.

Забезпечення цілісності і безпеки БД. Забезпечення цілісності БД складає необхідну умову успішного функціонування БД, особливо для випадку використання БД у мережі.

Цілісність БД є властивістю бази даних, яка означає, що в ній міститься повна, несуперечлива й адекватно відбиваюча предметну область інформація. Підтримка цілісності БД включає перевірку цілісності БД і її відновлення у випадку виявлення протиріч у базі даних. Цілісний стан БД описується за допомогою обмежень цілісності у виді умов, яким повинні задовольняти збережені в базі дані. Прикладом таких умов може служити обмеження діапазонів можливих значень атрибутів об'єктів, зведення про які зберігаються вБД, або відсутність повторюваних записів у таблицях реляційних БД.

Забезпечення безпеки досягається в системах керування базами даних шифруванням прикладних програм, даних, захисту паролем, підтримкою рівнів доступу до бази/даних і доокремих її елементів (таблиць, форм, звітів і т.ін.).

Контрольні питання до розділу 5

1. Надайте визначення поняттю «система керування базами даних».
2. Що визначає поняття «транзакція» стосовно баз даних?
3. У чому полягає призначення рівню зовнішніх моделей при організації систем керування базами даних за моделлю ANSI?
4. Перелічить етапи здійснення запиту користувача при роботі з системами керування базами даних.
5. Які дані зберігає база метаданих ?
6. Яких фахівців має містити група адміністратора баз даних ?
7. У чому полягає призначення серверів баз даних?
8. У чому полягає призначення буферизації при роботі з базами даних?

РОЗДІЛ 6. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ ТА НОРМАЛІЗАЦІЯ

6.1. Реляційна модель даних

В деяких випадках при ієрархічному і мережному представлені зростання бази даних може привести до порушення логічної організації даних. Такі ситуації виникають при появі нових користувачів, нових застосувань та видів запитів, при обліку інших логічних зв'язків між елементами даних. Недоліки ієрархічної і мережної моделей привели до появи нової, реляційної моделі даних, створеної Коддом у 1970 році. Реляційна модель була спробою спростити структуру бази даних і забезпечити незалежність представлення та опису даних від прикладних програм. У ній були відсутні явні покажчики на предків і нащадків, а всі дані були представлені у виді простих таблиць, розбитих на рядки і стовпці.

Реляційною називається база даних, у якій всі дані, доступні користувачу, організовані у вигляді таблиць, а всі операції над даними зводяться до операцій над цими таблицями. Для представлення реляційних баз даних розроблена формальна теорія баз даних, теоретичну основу якої складає алгебра та математична логіка. Основні взаємопов’язані поняття фізичного, спеціального прикладного та математичного рівнів побудови реляційної бази даних та їх взаємовідношення показані в таблиці 1, в рядках якої знаходяться еквівалентні поняття.

Таблиця 1 – **Основні поняття реляційної бази даних та їх взаємозв'язок**

	Фізичний рівень	**Спеціальний прикладний рівень**	**Математичний рівень**
1	Файл	Таблиця	Відношення
2	Запис	Рядок	Кортеж
3	Поле	Стовпець	Атрибут

У реляційній базі даних інформація організована у вигляді таблиць, розділених на рядки і стовпці, на перетині яких містяться значення даних. У кожної таблиці є унікальне ім'я, що описує її вміст. Масив значень, що можуть міститися в стовпці, називається доменом цього стовпця.

Така двохвимірна таблиця в математиці отримала назву відношення (relations).

Таблиця зрозуміла оглядово і звична для людей. Оригінальність підходу Е.Ф. Кодда полягає в застосовані до відношень впорядкованої системи операцій, що дозволяє отримувати складені відношення з бази (виводити, обчислювати, проводити арифметичні операції). Використання відношень дозволяє ділити інформацію на таку, що зберігається постійно і таку, яка отримується в результаті визначених перетворень над постійною. Е.Ф.Кодд показав, що набір відношень (таблиць) може бути використаний для збереження даних про об'єкти реального світу і моделювання зв'язків між ними. Під таблицею будеморозуміти структуру заголовку даної таблиці плюс сукупність записів даних у відповідності із заголовком. Наприклад, для збереження інформації про об'єктиз назвою «студент» можна використати відношення «СТУДЕНТИ» (Таблиця 2), в якому властивості об'єктів розміщуються в стовпцях таблиці.

Таблиця 2 – **Приклад відношення «СТУДЕНТИ»**

Прізвище І.Б.	**Дата народження**	**Курс**	**Напрямок**
Сидорчук Р.Д.	12.02.1990	5	Комп'ютерні науки
Петрук Д.В.	15.04.1992	3	Інформатика
Іванюк І.І.	18.11.1993	2	Прикладна математика
Федорук М.Ж.	31.11.1995	1	Комп'ютерні науки

У кожного стовпця в таблиці є своє ім'я (атрибут), що і служить заголовком стовпця. Список імен стовпців відношення називається схемою відношення. Усі стовпці в одній таблиці повинні мати унікальні імена, однак дозволяється привласнювати однакові імена стовпцям, розташованим в різних таблицях.

Таблиці не дозволяють проводити узгодження наступних трьох способів маніпулювання даними: впорядкування, групування по певних ознаках, доступ по дереву параметрів. Це пов'язано з тим, що в таблиці всі три способи

маніпулювання жорстко закріплені: дані впорядковані по одному параметру і невпорядковані по іншому.

Стовпці таблиці упорядковані зліва направо, й їхній порядок визначаєтьсяпри формуванні таблиці. У будь-якій таблиці завжди є як мінімум один стовпець.

Як правило, не вказується максимально допустиме число стовпців у таблиці, однак майже у всіх комерційних системах керування базами даних ця межа існує і, як правило, складає приблизно 255 стовпців.

На відміну від стовпців, рядки таблиці (кортежі) не мають визначеного порядку. Це значить, що якщо послідовно виконати два однакових запити для відображення вмісту таблиці, то немає гарантії, що обидва рази рядки будуть перераховані в тому самому порядку.

Рядки таблиці утворюють данні різного формату і різного типу, тобто можна стверджувати, що рядки таблиці є кортежами.

У таблиці може міститися будь-яка кількість рядків. Цілком припустиме існування таблиці з нульовою кількістю рядків. Така таблиця називається порожньою. Порожня таблиця зберігає структуру, визначену її стовпцями, просто в ній не містяться дані. Стандарти реляційних баз даних не накладають обмежень на кількість рядків у таблиці, і в багатьох системах керування базами даних розмір таблиць обмежений лише вільним дисковим простором комп'ютера.

Як правило, в сучасних реляційних БД допускається збереження символьних, числових даних, бітових рядків, спеціалізованих числових даних (таких як «грошовий»), а також спеціальних «темпоральних» даних (дата, час, часовий інтервал).

Найменша одиниця даних реляційної моделі – це окреме атомарне (неподільне) для даної моделі значення даних. Так, в одній предметній області прізвище, ім'я і по-батькові можуть розглядатися як єдине значення, а в інший –як три різноманітних значення.

Відношення називаються еквівалентними, якщо вони відрізняються тільки порядком чергування атрибутів.

Таблиці реляційної бази даних діляться на два класи: базові та розвідні.

Базова таблиця (каталоги предметної області). В реляційній базі даних базовою таблицею називається таблиця, яка включає один або декілька стовпців властивостей об’єкту і містить первинний ключ, що однозначно визначає цей об’єкт. Базова таблиця повинна містити первинний ключ. Базові таблиці часто називають первинними, оскільки вони мають первинний ключ.

Розвідні (проміжні) таблиці. Таблиця, що не є базовою, яка використовується для забезпечення зв’язків між іншими таблицями, називаєтьсяпроміжною таблицею. Ключові поля в проміжній таблиці повинні бути зовнішніми ключами, що зв’язані з первинними ключами базових таблиць. Проміжна таблиця може мати і інші атрибути, які функціонально залежать від цього зв’язку.

В реляційних базах даних використовуються спеціальні типи атрибутів

– первинні (унікальні) та вторинні (зовнішні) ключі.

Первинний ключ. Первинний ключ складається з набору значень, які однозначно визначають рядок (запис) базової таблиці. Будь-якому значенню первинного ключа повинен відповідати один і тільки один рядок (запис) таблиці. Первинний ключ, як правило, знаходиться у першому стовпці (№п/п). Інші атрибути функціонально залежать від даного ключа. Ключ може включатикілька атрибутів (складний ключ). Значення первинного ключа не повинні дублюватися. Це основне обмеження в реляційній базі даних для збереження цілісності даних.

Ключі-кандидати. Будь-який стовпець або група стовпців, які задовольняють вимогам, що накладаються на значення первинного ключа, є кандидатами на те, щоб стати первинним ключем.

Складені ключі. Якщо для виконання умов, що накладаються на значення первинного ключа, заданий ключ включає декілька полів таблиці, то тоді він називається складеним.

Вторинні (зовнішні) ключі. Зовнішній ключ – це стовпець, значення якого відповідають значенням первинного ключа з іншої зв’язаної таблиці. Він визначає посилання на конкретні записи інших таблиць, формуючи зв’язок між таблицями.

Умови і обмеження, які накладаються на відношення реляційних баз даних на табличному рівні представлення, можна сформулювати наступним чином:

- не може бути однакових первинних ключів, тобто всі рядки (записи)повинні бути унікальними;
- всі рядки повинні мати однакову типову структуру;
- імена стовпців в таблиці повинні бути різними, а значення стовпців
- повинні бути однотиповими;
- значення стовпців повинні бути атомарними, тобто не можуть бутикомпонентами інших відношень;
- повинна зберігатися цілісність для зовнішніх ключів;
- порядок розміщення рядків у таблиці неістотний – він впливає лише нашвидкість доступу до потрібного рядка.

Діаграма, на якій показано логічне представлення даних, називається схемою. Існує багато методів створення схем даних. Одним з найбільш розповсюджених є структурний метод схем даних (діаграми Бахмана).

Графічний опис структури таблиць у формі полосок, які містять імена полів і показують спрощені відношення між даними, використовується для того,щоб користувачам було легше зрозуміти розроблену модель даних.

Іншим розповсюдженим підходом є метод, в якому використовується схема “Елемент – Відношення” (E-R), яка була розроблена Пітером Ченом в 1976р.. E-R схеми призначені для наглядного представлення відношень між об’єктами і поведінки елементів.

Елементи даних вказані у прямокутниках, атрибути даних – в овалах, а відношення між елементами – в ромбах. Відношення між об’єктами бази даних

на концептуальному етапі можуть визначатися їх поведінкою.

Таким чином E-R схеми включають принаймні одне дієслово, об'єкт якого знаходиться справа від символу відношення. Символи наносяться на схему по мірі конкретизації моделі. Однією з переваг E-R схем є те, що їх можна використовувати для представлення на порівняно невеликому просторі концептуальної моделі великих схем з багатьма базами даних.

Наступними розглянемо типи відношень. Так, відношення "один-до-одного" є найпростішим відношенням між таблицями є відношення "один-до-одного". В такому відношенні одному запису однієї таблиці відповідає тільки один запис у іншій. Таблиці, що зв'язані відношенням "один-до-одного" можна об'єднати в одну таблицю, яка складається з полів обох таблиць. Відношення "один-до-одного" часто використовують для розділення таблиць, що містять велику кількість полів. Наприклад, це може бути потрібним для того, щобскоротити час перегляду полів, що містять певний набір даних. В деяких випадках необхідно керувати доступом до частин таблиць, які містять важливі або конфиденційні дані. На рис. 3 показана E-R схема для таблиці "Інженер" та "Комп'ютер". Одиниці з обох сторін ромба вказують на відношення "один-до одного".

Відношення "один-до-багатьох" Відношення "один-до-багатьох" зв'язує один запис першої таблиці з декількома записами другої за допомогою первинного ключа базової таблиці і відповідного йому зовнішнього ключа зв'язаної таблиці. Зовнішній ключ таблиці, що містить велику кількість відношень, може входити до складеного первинного ключа, але він є зовнішнім по відношенню до базової таблиці. Відношення "один-до-багатьох" використовується найбільш часто. На E-R схемі, що показана на рис.4 , це відношення позначено символом ∞.

Відношення "багато-до-одного" протилежно відношенню "один-до-багатьох".

Якщо вибір відношення "багато-до-одного" або "один-до-багатьох" не маєвеликої ролі, то відношення між таблицями називається рефлексивним. Відношення "багато-доодного" є відображенням відношення "один-до-багатьох". Всі відношення "багато-доодного" в Access є рефлексивними. На E-Rсхемі рефлексивні відношення позначаються дієсловом у відповідній формі, який розміщується зовні ромба, що визначає відношення.

Наступним буде відношення "багато-до-багатьох". У випадку, коли відсутнє забезпечення цілісності зв'язків між таблицями, можна говорити про відношення "багато-до-багатьох". Оскільки забезпечення цілісності зв'язків міжтаблицями є базовою вимогою реляційної моделі даних, то відношення "багато- добагатьох" не використовуються, а при зустрічанні уникається введенням розвідної таблиці.0

Сучасні бази даних можуть представлятися сукупністю взаємопов'язаних таблиць. Тіло типового файлу бази даних для однієї таблиці містить заголовок таблиці та власне послідовно організований набір записів. В заголовку файлу міститься наступна інформація:

1) перелік полів з характеристиками (назва, тип, довжина поля);
2) час створення чи обновлення файлу;
3) кількість записів, що знаходяться у файлі,
4) інша допоміжна інформація (наприклад, інформація про зв'язок з індексним файлом, відомості про мову заповнення та інше).

Кожен запис таблиці складається із окремих полів (атрибутів), які діляться на три групи: ключові, вказівні та допоміжні. Ключовими є ті атрибути, якими однозначно ідентифікується даний запис. Двох записів з однаковими ключовими атрибутами бути не може. Вказівні атрибути виконують роль ключових атрибутів в інших базах даних, на які посилається даний запис. З їх допомогою можна отримати додаткову інформацію для заданого запису. Допоміжні атрибути – це характеристики для заданого запису і вони, як правило, можуть повторюватись.

Файл бази даних може бути індексованим або ні. Наявність індексації означає, що всі записи в файлі баз даних перевпорядковані у відповідності із наперед заданим принципом (алфавітний порядок, порядок зростання чи спадання і так далі) по одному з полів. Інформація про індексацію міститься в індексному файлі у вигляді набору взаємопов'язаних пар чисел. Система індексації застосовується для зручності використання бази даних: для заданих полів створюється впорядкований перелік пар чисел, перше з яких вказує фізичний реальний порядковий номер запису в файлі, друге – порядковий номер даного запису у перевпорядкованому списку. Розрізняють так звані одиничні і множинні індексні файли. Для кожного фізичного запису одиничний індексний файл містить тільки одну пару чисел, в той час як множинний – множину з декількох впорядкованих пар чисел. Кожне друге з наступної пари чисел в наборі означає порядковий номер у новому списку. Характеристикою типу інформації даного атрибуту є тип поля. Найчастіше вживані наступні позначення типів полів, які наводяться в таблиці 3.

Таблиця 3 – **Типи полів файлів баз даних**

№	Позначення	Тип поля
1	**N**	Числовий
2	**F**	Числовий з плаваючою комою
3	**C**	Символьний
4	**D**	Дати/час
5	**M**	Мемо-поле
6	**L**	Логічний

В деяких сучасних системах керування базами даних введено нові типи полів, але вони або надають нові можливості (наприклад тип графічного об'єкту) або просто розширюють наведені типи полів. Крім цього, для кожного числового чи символьного поля задається його довжина в символах. Так, для поля дати відведено 8 позицій, а для логічного – одне (логічний нуль – "false" або одиниця – "true"). Поле мемо містить

стандартної довжини вказівник на текстову інформацію. Ці вказівники містять адреси послідовно організованих записів в допоміжному файлі.

6.2. Тринадцять правил КОДДА для реляційних систем керування базами даних

Кодд у 1985 році сформулював 13 правил (0..12), яким повинна задовольняти будь-яка СКБД, що претендує на тип реляційної. З того часу дванадцять правил Кодда вважаються визначенням реляційної СКБД.

Тринадцять правил Кодда, яким повинна відповідати реляційна СКБД, є наступними.

0. Фундаментальне правило (Foundation Rule) Реляційна СКБД повинна бути здатна повністю управляти базою даних через її реляційні можливості.

1. Правило інформації (Information Rule) Вся інформація в базі даних

повинна бути надана винятково на логічному рівні і тільки одним способом – увигляді значень, що містяться в таблицях.

2. Правило гарантованого доступу (Guaranteed Access Rule) Логічний доступ до всіх і кожного елементу даних (атомарному значенню) у реляційній базі даних повинний забезпечуватися шляхом використання комбінації імені таблиці, первинного ключа та імені стовпця. Правило 2 вказує на роль первинних ключів при пошуку інформації в базі даних. Ім'я таблиці дозволяє знайти необхідну таблицю, ім'я стовпця дозволяє знайти необхідний стовпець, а первинний ключ дозволяє знайти рядок, який містить шуканий елемент даних.

3. Правило підтримки недійсних значень (Systematic Treatment of Null Values)

У реляційній базі даних повинна бути реалізована підтримка недійсних значень, що відрізняються від рядка символів нульової довжини, рядка символівпробілів чи нуля будь-якого іншого числа і використовуватися для представлення відсутніх даних незалежно від типу цих даних. Правило 3 вимагає, щоб відсутні дані можна було представити за допомогою недійсних значень (NULL).

4. Правило динамічного каталогу, заснованого на реляційній моделі (Dynamic Online Catalog Based on the Relational Model) Опис бази даних на логічному рівні необхідно представити в тому ж вигляді, що й основні дані, щобкористувачі, які володіють відповідними правами, могли працювати з ним за допомогою тієї ж реляційної мови, яку вони застосовують для роботи з основними даними. Правило 4 говорить, що реляційна база даних повинна сама себе описувати. Іншими словами, база даних повинна містити набір системних таблиць, що описують структуру самої бази даних.

5. Правило вичерпної підмови даних (Comprehensive Data Sublanguage Rule) Реляційна система може підтримувати різні мови і режими

взаємодії з користувачем (наприклад, режим питань і відповідей). Однак повинна існувати принаймні одна мова, оператори якої підтримують наступні елементи: визначення даних; визначення представлень; обробку даних (інтерактивну і програмну); умови цілісності; ідентифікацію прав доступу; границі транзакцій (початок, завершення і скасування).

Правило 5 вимагає, щоб СКБД використовувала мову реляційної бази даних, наприклад SQL, хоча явно SQL у правилі не згадають. Така мова повинна підтримувати всі основні функції СКБД – створення бази даних, читання і введення даних, реалізацію захисту бази даних і т.д..

6. Правило відновлення представлень (View Updating Rule) Усі представлення, які теоретично можна обновити, повинні бути доступні для відновлення. Правило 6 дозволяють показувати різним користувачам різні

фрагменти структури бази даних.

7. Правило додавання, відновлення і вилучення (High-level Insert,/Update/Delete) Можливість працювати з відношенням як з одним операндом повинна існувати не тільки при читанні даних, але і при додаванні, відновленні і вилученні даних. Правило 7 акцентує увагу на тому, що бази даних по своїй природі орієнтовані на множини. Воно вимагає, щоб операції додавання, видалення і відновлення можна було виконувати над множинами рядків. Це правило призначене для того, щоб заборонити реалізації, у яких підтримуються тільки операції над одним рядком.

8. Правило незалежності фізичних даних (Physical Data Independence) Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при будь-яких змінах методів збереження даних чи методів доступу до них.

9. Правило незалежності логічних даних (Logical Data Independence) Прикладні програми й утиліти для роботи з даними повинні на логічному рівні залишатися недоторканими при внесенні в базові таблиці будь-яких змін, які теоретично дозволяють зберегти недоторканими дані, що містяться в цих таблицях. Правила 8 і 9 стверджують, що конкретні способи реалізації чи збереження доступу, які використовуються в СКБД, і навіть зміни структури таблиць бази даних не повинні впливати на можливість користувача працювати з даними.

10. Правило незалежності умов цілісності (Integrity Independence)

Повинна існувати можливість визначати умови цілісності, специфічні для конкретної реляційної бази даних, на підмові реляційної бази даних і зберігати їх у каталозі, а не в прикладній програмі. Правило 10 вимагає, щоб мова бази даних підтримували обмежувальні умови, які накладаються на дані, що вводяться, і дії, що можуть бути виконані над даними.

11. Правило незалежності поширення (Distribution Independence) Реляційна СКБД не повинна залежати від потреб конкретного клієнта. Правило 11 говорить, що мова бази даних повинна забезпечувати можливість роботи з розподіленими даними, розташованими на інших комп'ютерних системах.

12. Правило одиничності (The Nonsubversion Rule) Якщо в реляційній системі є низькорівнева мова (що обробляє один запис за один раз), то повинна бути відсутня можливість використання її для того, щоб обійти правила й умовицілісності, виражені на реляційній мові високого рівня (що обробляє кілька записів за один раз).

Правило 12 запобігає використання інших можливостей для роботи з базою даних, крім мови бази даних, оскільки це може порушити її цілісність.

Жодна з сучасних систем керування базами даних не дотримується цих правил від і до. Причина цього – складність правил.

Згідно Дейту, реляційна модель складається з трьох частин, що описують різні аспекти реляційного підходу: структурної частини, маніпуляційної частини та і цілісній частини. У структурній частині моделі фіксується, що єдиною структурою даних, яка використовується в реляційних БД, є нормалізоване n-арне відношення. У маніпуляційній частини моделі стверджується два фундаментальних механізми маніпулювання реляційними БД – реляційна алгебра і реляційне числення. Перший механізм базується в основному на класичній теорії множин (з деякими уточненнями), а другий – на класичному логічному апараті числення предикатів першого порядку.

У цілісній частині реляційної моделі даних фіксуються дві базових вимоги цілісності, що повинні підтримуватися в будь-якій реляційній системі керування базами даних.

Перша вимога називається вимогою цілісності сутностей. Об'єкту або сутності реального світу в реляційній БД відповідають кортежі відношень. Конкретно вимога полягає в тому, що будь-який кортеж будь-якого відношеннявідрізняється від будь-якого іншого кортежу цього відношення, тобто іншими словами, будь яке відношення повинно мати первинний ключ.

Друга вимога називається вимогою цілісності по посиланнях.

Очевидно, що при дотриманні нормалізованості відношень складні сутності реального світу представляються в реляційній БД у виді декількох кортежів декількох відношень

Вимога цілісності по посиланнях, або вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, який з'являється у відношенні, до якого посилаються, повинний існувати кортеж із таким же значенням первинного ключа. Так, наприклад, якщо для співробітника зазначений номер відділу, то цей відділ повинен існувати.

Степінь відношення – це число його атрибутів. Відношення степеня один називають унарним, степеня два – бінарним, ступеня три – тринарним, ... , а степеня n – n-арним.

Кардинальне число або потужність відношення – це число його кортежів. Кардинальне число відношення змінюється в часі на відміну від його степені.

До переваг реляційної бази можна віднести: незалежність від фізичного рівня представлення; зручність і розуміння організації даних користувачами; максимальна гнучкість при обробці непередбачених запитів; можливість

розширення бази приєднанням нових елементів, записів беззміни при цьому існуючих підсхем та прикладних програм.

6.3. Нормалізація даних в реляційній моделі

Практика розробки та експлуатації баз даних викристалізувала базові вимоги до їх побудови. Це зокрема вимоги: цілісності даних (обмеження; правила використання обмежень; правила обробки при порушенні обмежень цілісності; ефективність використання обмежень); узгодженості даних; відновлюваності даних (при збоях обладнання); безпеки (від несанкціонованих дій); ефективності (яка визначається такими параметрами, як швидкодія, мінімальний час доступу та мінімальна пам’ять).

З іншої сторони, в середовищі прикладних користувачів-розробників булисформовані так звані бізнес-правила для роботи з базами даних. Їх суть полягає в ефективній реалізації роботи з даними, яка гарантує, що дані, які містяться в базах даних, повинні відповідати політиці й стратегії організації, зокрема і її правилам. Наприклад, для реалізації цього принципу потрібно встановити вимоги до кожного стовпця в таблиці баз даних так, щоб він не допускав значень, які суперечили б політиці організації. Фактично бізнес-правила вимагають виконання трьох основних функцій: зберігати небажані дані поза базами даних; однозначно описувати і строго визначати зв’язки між стовпцями ітаблицями; при необхідності надавати інформацію звідки, коли і ким дані внесені вбазу даних.

В першу чергу, реляційні бази даних повинні задовольняти так званим умовам нормалізації. Процес трансформації даних в реляційну форму називається нормалізацією баз даних. Нормалізація – це видалення надлишкових даних з кожної таблиці бази даних. Процес нормалізації переслідує подвійну мету: видалити надлишкові копії даних і забезпечити максимальну гнучкість, як в структурах таблиць, так і в її інтерфейсних програмах на випадок можливих майбутніх змін баз даних. Найчастіше зустрічаються п’ять форм нормалізації, що означає п’ять різних установок реляційного критерію нормалізації баз даних.

1. Перша нормальна форма. Для першої нормальної форми потрібно, щоб таблиця була двовимірною і не містила груп, що повторюються. Вона не повинна містити комірок, що включають кілька значень (атомарність, неподільність.) Поле вважається неподільним, якщо воно містить тільки один елемент даних.

2. Друга нормальна форма. Перед перевіркою на відповідність другій нормальній формі таблиця повинна бути приведена до першої нормальної форми. Для другої нормальної форми потрібно, щоб дані у всіх не ключових стовпцях повністю залежали від первинного ключа. Під повною залежністю розуміється те, що значення в кожному не ключовому стовпці однозначно визначається значенням первинного ключа. Якщо одне з полів не залежить від величини первинного ключа, то необхідно включити в ключ доповнювальні таблиці. Друга нормальна форма дозволяє видалити більшу

частину даних, що повторюються, які часто залишаються після першого етапу нормалізації.

3. Третя нормальна форма. Потрібно, щоб таблиці були попередньо приведені до першої та другої нормальної форми. Для третьої нормальної форми потрібно, щоб всі неключові стовпці таблиці не тільки залежали від первинного ключа таблиці, але й були незалежними один від одного, тобто, щоб були відсутні транзитивні функціональні залежності між стовпцями таблиці.

4. Четверта нормальна форма. Таблиця має задовольняти вимогам третьої нормальної форми і, крім того, забороняється зберігати незалежні елементи в одній і тій же таблиці, коли між цими елементами існує зв'язок (зв'язки) багато до багатьох.

5. П'ята нормальна форма має місце таблиця задовольняє вимогам четвертої нормальної форми та існує можливість модернізації даних таблиці. Передбачається, що при розробці баз даних і особливо при проведенні нормалізації таблиць потрібно звертати велику увагу на те, щоб випадково не проігнорувати суттєву характеристику або параметри об'єкта. Нормалізація збільшує число відношень в базах даних і тим самим час обробки. Але за рахунок коректності і усунення дублювання відбувається прискорення виконання доступу до даних.

6.4. Алгебра відношень

Ефективність реляційної моделі бази даних визначається здатністю виконувати над відношеннями наступні операції алгебри відношень: об'єднання, перетин, різниця, декартовий добуток, ділення, проекція, вибір, з'єднання. Так, ступінь відношення – це кількість атрибутів, які в нього входять (або кількість стовпців у таблиці). Потужність відношення – кількість записів у таблиці відношень (або кількість рядків без заголовку в таблиці).

Операція об'єднання проводиться над двома відношеннями. Результуюче відношення включає всі записи першого відношення і ті записи другого відношення, яких немає в першому.

Перетин виконується над двома відношеннями. Результуюче відношення містить тільки ті записи, які є одночасно в першому і другому відношеннях.

Операція різниці проводиться над двома відношеннями. Результуюче відношення містить ті записи першого відношення, яких немає в другому відношенні.

Декартовий добуток виконується над двома відношеннями, степінь результуючого відношення дорівнює сумі степенів первинних відношень, а потужність рівна добутку їх потужностей. Результуюче відношення містять всі можливі комбінації в записі первинних відношень.

Операція ділення – відношення дільника повинно містити підмножину атрибутів відношення діленого. Результуюче відношення включає тільки ті записи декартового добутку результуючого відношення з дільником, які

містяться в діленому. Крім того, результуюче відношення містить тільки ті відношення діленого, яких немає в дільнику.

Операція проекції виконується над одним відношенням. Результуюче відношення включає частину атрибутів вихідного, на які виконується проекція.

Контрольні питання до розділу 6

1. Які особливості має реляційна база даних ?
2. У чому полягає сутність інфологічної моделі даних?
3. Поясніть принцип відношень “один-до-одного” у реляційних базах.
4. Поясніть принцип відношень “один-до-багатьох” у реляційних базах.
5. Поясніть зміст поняття «індексований файл» бази даних.
6. Поясніть правило «гарантованого доступу» з системи правил Кодда.
7. Охарактеризуйте п’ять нормальних форм реляційних баз даних.

РОЗДІЛ 7. ЗАГАЛЬНА ХАРАКТЕРИСТИКА БАЗ ЗНАНЬ

7.1. Базові поняття щодо баз знань

Знання пов'язані з даними, базуються на них, але представляють результатрозумової діяльності людини, узагальнюють його досвід, отриманий в ході виконання якої-небудь практичної діяльності. Вони є результатом емпіричного досвіду. Знання - це виявлені закономірності предметної області (принципи, зв'язки, закони), що дозволяють вирішувати завдання в цій галузі. При обробці на ЕОМ знання трансформуються аналогічно даними:

- знання в пам'яті людини як результат мислення;
- матеріальні носії знань (підручники, методичні посібники);
- поле знань - умовне опис основних об'єктів наочної області, їх атрибутів і закономірностей, які їх пов'язують;
- знання, описані на мовах подання знань (продукційні мови, семантичні мережі, фрейми);
- бази знань.

Часто використовуються такі визначення знань: знання - це добре структуровані дані, або дані про дані, або метадані.

Існує безліч образів визначати поняття. Один з широко вживаних образів, заснований на ідеї інтенсіонал. Інтенсіонал поняття – це визначення через поняття більш високого рівня абстракції із зазначенням специфічних властивостей. Цей образ визначає знання. Інший спосіб визначає поняття через перерахування понять нижчого рівня ієрархії або фактів, які відносяться до визначеного. Це є визначення через дані, або екстенсіонал поняття.

Знання можуть бути класифіковані за такими категоріями:

- поверхневі - знання про видимі взаємозв'язки між окремими подіями

і фактами у наочній області;

- глибинні - абстракції, аналогії, схеми, які відображають структуру і процеси у наочній області.

Знання можна розділити на процедурні та декларативні. Декларативні знання – це знання, які записані в пам'яті інтелектуальної системи так, що вони безпосередньо доступні для використання після звернення до відповідного полю пам'яті. Зазвичай декларативні знання використовуються для подання інформація про властивості та факти предметноїобласті. За формою подання декларативні знання протиставляються процедурним знанням.

Процедурні знання – це знання, що зберігаються в пам'яті інтелектуальної системи у вигляді описів процедур, за допомогою яких їх можна отримати. Зазвичай процедурні знання використовуються для представлення інформації про способи вирішення завдань в проблемній області.Це також різні інструкції, методики тощо.

Історично первинними були процедурні знання, тобто знання, "розкриті"в алгоритмах. Вони управляли даними. Для їх зміни необхідно було міняти програми. Однак з розвитком штучного інтелекту пріоритет даних поступово змінювався, і все більша частина знань зосереджувалася в структурах даних (таблиці, списки, абстрактні типи даних), тобто збільшувалася рольдекларативних знань.

Сьогодні знання придбали чисто декларативну форму, тобто знаннями вважають пропозиції, записані на мовах подання знань, наближених до природних і зрозумілим неспеціалістам. Існують десятки моделей (або мов) подання знань для різних предметних областей. Більшість з них може бути зведені до наступних класів: продукційні; семантичні мережі; фрейми; формальні логічні моделі та багато інших.

7.2. Стратегії отримання знань

Розрізняють кілька стратегій отримання знань. Найбільш поширені – такі: придбання; витягування; формування.

Під придбанням знань розуміється спосіб автоматизованого побудови бази знань за допомогою діалогу експерта і спеціальної програми (при цьому структура знань заздалегідь закладається в програму). Ця стратегія потребує суттєвого попередньої обробки предметної області. Системи набуття знань дійсно купують готові фрагменти знань згідно структурам, закладеним розробникам систем. Більшість цих інструментальних засобів орієнтована на конкретні експертні системи з жорстко визначеною предметною областю і моделлю представлення знань, тобто не є універсальними.

Термін витягування знань стосується безпосереднього живого контакту інженера по знаннях і джерела знань. Автори схильні використовувати цей термін як більш ємний і такий, що більш точно відображає суть процедури перенесення компетентності експерта через інженера по знаннях в базу знань експертної системи.

Термін формування знань традиційно закріпився за надзвичайно

перспективною галуззю інженерії знань, яка займається розробкою моделей, методів і алгоритмів аналізу даних для отримання знань і навчання, які активно розвиваються. Ця область включає індуктивні моделі формування гіпотез на основі повчальних вибірок, вчення аналогічно і інші методи.

7.3. Висновки на базі знань

База знань – сукупність знань, які відносяться до деякої предметної області, формально представленим так, щоб на них основі можна було здійснювати роздуми. Бази знань найчастіше використовуються в контексті експертних систем, де з їх допомогою подаються навички та досвід експертів, зайнятих практичною діяльністю у відповідній галузі (наприклад, в медицині або в математиці). Зазвичай база знань є сукупністю правил виведення.

Не дивлячись на всі недоліки, найбільше поширення набула продукційна модель. При використанні продукционной моделі база знань складається з набору правил. Програма, яка управляє перебором правил, називається машиною виведення. Машина виведення (інтерпретатор правил) виконує дві функції: по-перше, перегляд існуючих фактів з робочої пам'яті (бази даних) і правил з бази знань і додавання (в міру можливості) в робочу пам'ять нових фактів, а по-друге, визначення порядку перегляду і вживання правил. Цей механізм управляє процесом консультації, зберігаючи для користувача інформацію про отримані висновки, і запрошує у нього інформацію, коли для спрацьовування чергового правила в робочій пам'яті виявляється недостатньо даних

У переважній більшості систем, заснованих на знаннях, механізм виведення є невеликою за обсягом програмою і включає два компонента: один реалізує власно висновок, інший керує цим процесом. Дія компонента виведення ґрунтовано на вживанні правила, яке називається modus ponens.

Правило modus ponens. Якщо відомо, що достеменно твердження А, то існує правило виду - «Якщо А, то В», тоді твердження У той же істинно. Правила спрацьовують, коли знаходяться факти, які задовольняють їх лівій частині: якщо посилання істинна, то повинен бути справжній і висновок. Компонент виведення повинен функціонувати навіть при нестачі інформації. Отримане рішення може і не бути точним, проте, система не повинна зупинятися через те, що відсутня будь-яка частина вхідної інформації. Компонент, який управляє, визначає порядок вживання правил і виконує чотири функції:

1. Зіставлення зразок правила зіставляється з наявними фактами.
2. Вибір якщо в конкретній ситуації може бути застосовано відразу кількаправил, то з них вибирається одне, найбільш відповідне по заданому критерію (рішення конфлікту).
3. Спрацювання якщо зразок правила при зіставленні збігся з якимись фактами з робочої пам'яті, то правило спрацьовує.
4. Дія робоча пам'ять піддається зміні шляхом додавання в неї виведення спрацьованого правила. Якщо в правій частині правила міститься

вказівка на будь-яку дію, то воно виконується (як, наприклад, в системах забезпечення безпеки інформації).

Інтерпретатор продукцій працює циклічно. У кожному циклі він переглядає всі правила, щоб вказати ті, посилання яких збігаються з відомими на даний момент фактами з робочої пам'яті, Після вибору правило спрацьовує, його висновок заноситься в робочу пам'ять, і потім цикл повторюється спочатку. В одному циклі може спрацювати лише одне правило. Якщо кілька правил успішно зіставлені з фактами, то інтерпретатор виконує вибір за певним критерієм єдиного правила, яке спрацьовує в даному циклі. Цикл роботи інтерпретатора схематично представлений на рис. 7.1.

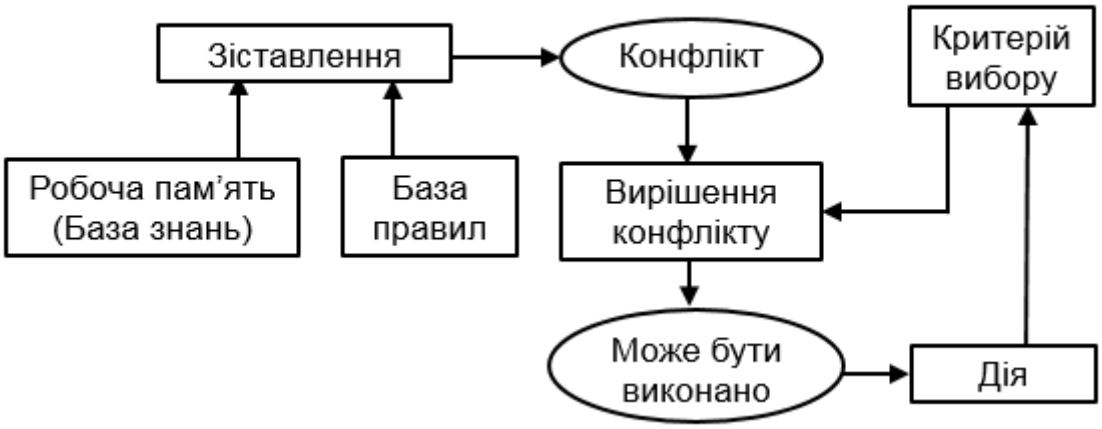

Рис. 7.1. Цикл роботи інтерпретатора

Інформація з робочої пам'яті послідовно зіставляється з посиланнями правил для виявлення успішного зіставлення. Сукупність відібраних правил складає так зване конфліктне безліч. Для вирішення конфлікту інтерпретатор має критерій, за допомогою якого він обирає єдине правило, після чого правило спрацьовує. Це виражається в занесенні фактів, які створюють висновок правила, в робочу пам'ять або в зміні критерію вибору конфліктуючих правил. Якщо ж по закінченню правила згадується назва якого-небудь дії, то воно виконується. Робота машини виведення залежить лише від стану робочої пам'яті і від стану бази знань. На практиці зазвичай враховується історія обробки, тобто поведінку механізму випаду в попередніх циклах. Інформація про поведінку механізму виведення запам'ятовується в пам'яті станів. Зазвичай пам'ять станів містить протокол системи.

Від обраного методу пошуку, тобто стратегії виведення, буде залежати порядок вживання і спрацьовування правил. Процедура вибору зводиться до визначення напрямку пошуку і способу його здійснення. Процедури, які реалізовують пошук, зазвичай закладені в механізм виведення, тому в більшостісистем інженери знань не мають до них доступу і, отже, не можуть у них нічого міняти за власним бажанням. При розробці стратегії управління виводу поважно визначити два питання:

1. Яку точку в просторі станів прийняти як початкову? Від вибору цієї

точки залежить і метод здійснення пошуку – в прямому або зворотному напрямку.

2. Якими методами можна підвищити ефективність пошуку рішення?

Ці методи визначаються обраною стратегією перебору - глибину, ширину, по підзадач або ін. При зворотному порядку виведення спочатку висувається деяка гіпотеза, а потім механізм виведення як би повертається назад, переходячи до фактів, намагаючись знайти ті, які підтверджують гіпотезу. Якщо вона виявилася правильною, то вибирається наступна гіпотеза, яка деталізує першу і є по відношенню до неї підцілі. Далі відшукуються факти, що підтверджують істинність підпорядкованої гіпотези. Висновок такого типу називається керованим цілями, або керованим консеквента. Зворотний пошук застосовується в тих випадках, коли цілі відомі їїх порівняно небагато.

7.4. Елементи експертних систем

Експертна система - це комплекс комп'ютерного програмного забезпечення, яке допомагає людині приймати обґрунтовані рішення. Експертні системи використовують інформацію, отриману заздалегідь від експертів - людей, які в якій-небудь області є кращими фахівцями. Всі експертні системи включають, принаймні, три основні елементи: базу знань, машину висновку й інтерфейс користувача (рис. 7.2).

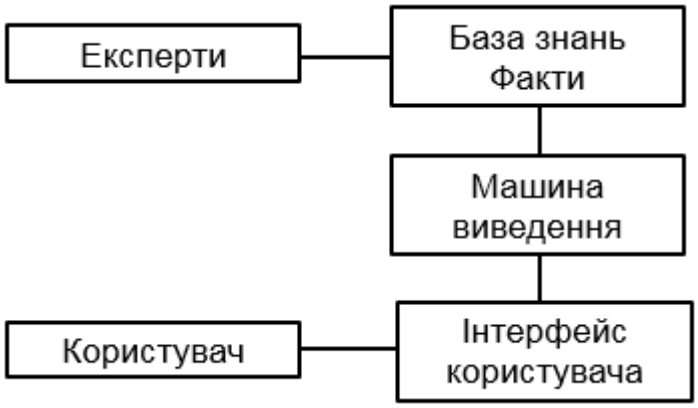

Рис. 7.2. Елементи експертної системи

База знань містить відомі факти, виражені у вигляді об'єктів, атрибутів і умов. Крім описових уявлень про дійсність, вона включає вираження невизначеності - обмеження на достовірність факту. В цьому відношенні вона відрізняється від традиційної бази даних внаслідок свого символьного, а не числового або літерного вмісту. При обробці інформації бази даних користуються заздалегідь певних логічних правил Відповідно, база знань, яка представляє вищий рівень абстракції, має справу з класами об'єктів, а не з самими об'єктами. База знань створюється людьми - консультантами.

Головним в експертній системі є механізм, який здійснює пошук в базі знань з правил раціональної логіки для отримання рішень. Ця машина виведення наводиться в дію при отриманні запиту користувача і виконує такі

завдання: порівнює інформацію, яка міститься в запиті користувача, з інформацією бази знань; шукає певну мету або причинні зв'язку; оцінює відносну визначеність фактів, спираючись на відповідні коефіцієнти довіри, пов'язані з фактом.

Як випливає з її назви, машина виведення призначена для побудови висновків. Її дія аналогічно міркуванням експерта-людини, який оцінює проблему і пропонує гіпотетичні рішення. У пошуку цілей на основі запропонованих правил, машина виведення звертається до бази знань до тих пір, поки не знайде достовірний шлях до отримання прийнятного результату.

Завдання інтерфейсу користувача полягає в організації обміну інформацією між оператором і машиною виведення. Інтерфейс з використанням природної мови створює видимість довільної бесіди, застосовуючи повсякденні вирази в правильно побудованих пропозиціях. Очевидно, що чим природніше такий інтерфейс, тим вище вимоги до зовнішньої і оперативної пам'яті.

Отже, системи, які надають користувачеві максимум зручностей, витрачають більше ресурсів основної машини, доступних з віддалених робочих станцій. Інструментальні засоби, призначені для роботи на персональних комп'ютерах, які мають обмежені можливості, неминуче приносять "дружність"в жертву ефективності.

Інтерфейс прямого введення повинен вміти розпізнавати мову, або, принаймні, достатня кількість ключових слів і фраз, щоб вловлювати їх зв'язок з даною проблемою і її пропонованими рішеннями.

У роботі з експертними системами беруть участь як мінімум три групи людей. По-перше, адміністрація встановлює призначення експертної системи, обмежує предметну область, яку повинна охоплювати система, і точно визначає, які вигоди організація зможе отримувати з її використання. По-друге, фахівець з збору знань (інженер - когнитологией) збирає інформацію, необхідну для бази знань, порівнює відповідні данини і евристичний організовує інформацію. По-третє, потенційний користувач вказує, як буде використовуватися система, якого роду проблеми доведеться вирішувати, і яким чином буде здійснюватися взаємодія програми з оператором.І, врешті-решт, системі потрібен експерт (частіше група експертів) в певній предметній встановленої наочної області, для отримання від нього знань, як в формі фактичної інформації, так і щодо аналітичних методів, які застосовуються для вирішення проблем в цій галузі.

Комп'ютерну частина системи представляютькомпоненти програмного забезпечення, які обробляють отриману інформацію про дійсність, закладеної в символьному вигляді в базу знань. У базу знань надходять факти. Зв'язок між фактами представлена евристичними правилами - виразами декларативного знання про відносини між об'єктами. Кожне таке правило має складову "якщо і компонент" то "(висновок-дія), які визначають пряма або зворотна причинно-наслідковий зв'язок. Дійсні затвердження лише достовірні, іншими словами, міра їх визначеності не завжди абсолютна.

Кількісні коефіцієнти визначеності збільшують точність міркування експертних систем. Загальноприйнята схема полягає в тому, щоб варіювати рівень довіри від 0, що представляє мінімальну міру визначеності, до 100 - вищаміра.

Контрольні питання до розділу 7

1. Назвіть основні перетворення знань при машинній обробці.
2. Поясніть поняття інтенсіонал і екстенсіонал поняття.
3. Наведіть класифікацію знань.
4. Надайте визначення бази знань.
5. Як ви розумієте правило modus ponens?
6. Які стратегії управління виводу ви знаєте?
7. Дайте визначення поняття «Експертна система».

РОЗДІЛ 8. CASE-ЗАСОБИ ДЛЯ ПОБУДОВИ ДІАГРАМ UML

8.1. IBM Rational Rose

Rational Rose - це сучасний та потужний інструмент для аналізу, моделювання та розробки програмних систем. Він ефективно використовується для вирішення різних задач проектування інформаційних систем: від аналізу бізнес-процесів до генерації коду на конкретній мові програмування. Цей інструмент дозволяє як проектувати нові системи, так і вдосконалювати існуючі за допомогою зворотного проектування.

Для того, щоб повністю покрити весь ринок засобів проектування та розробки, випускаються кілька версій продукту:

1. Rational Rose Modeler: Ця версія дозволяє аналітикам і проектувальникам проводити аналіз бізнес-процесів та проектувати системи. Вона не підтримує генерацію коду.
2. Rational Rose Professional: Професійна редакція продукту, яка залежно від вибраної мови програмування, дозволяє виконувати пряме і зворотне проектування. Замовляється у визначених конфігураціях, таких як Rose Professional C++ або Rose Professional C++ DataModeler. Продукт генерує каркасний код інформаційної системи на вибраній мові, який потребує подальшого програмування.
3. Rational Rose RealTime: Версія, спеціально створена для генерації 100% виконуваного коду в режимі реального часу, з підтримкою мов C або C++. За твердженням розробників, модель автоматично компілюється та збирається у виконуваний файл.
4. Rational Rose Enterprise: Найповніша версія, що підтримує всі функції інших редакцій, за винятком можливості 100% генерації коду. Вона охоплює весь спектр завдань з проектування, аналізу та генерації коду, будучи універсальним програмним пакетом для всіх учасників проекту.
5. Rational Rose DataModeler**:** Функціональність з проектування баз

даних, яка входить до складу Rose Enterprise або Professional.

Безкоштовна версія продукту відсутня, але для освітніх установ програмне забезпечення IBM доступне безкоштовно в рамках програми IBM Academic Initiative. Залежно від постачання, Rational Rose може мати різні набори візуальних компонентів (діаграм). Однак, продукт і так достатньо функціональний:

1. Пряме і зворотне проектування на мовах: ADA, Java, C, C++, Basic.
2. Підтримка технологій COM, DDL, XML.
3. Генерація схем БД Oracle і SQL.

8.2. Borland Together

Borland Together ControlCenter - це інтегрована платформа розробки, яка спрощує і прискорює аналіз, дизайн, розробку та розгортання складних корпоративних застосувань. Об'єднуючи всі можливості в одному інтегрованому рішенні з підтримкою UML, Borland Together сприяє командній розробці високоякісних систем швидше і ефективніше. Основні особливості Borland Together:

1. Підтримка XP (екстремальне програмування**)**: Together підтримує гнучкі процеси моделювання, надаючи інтерактивні можливості моделювання та підтримуючи всі види діаграм UML.
2. Прискорення процесів розробки за допомогою патернів**:** використання шаблонів проектування дозволяє швидко створювати моделі відповідно до корпоративних стандартів і кращих практик кодування.
3. Розгортання додатків на кілька серверів без перекодування**:** додаток можна легко розгорнути на різних серверах додатків, просто додавши кілька рядків коду.
4. Функція контролю якості**:** вбудоване функціональне тестування допомагає виявляти проблеми на ранніх етапах розробки, що значно знижує вартість виправлення помилок.

Borland Together - це набагато більше, ніж просто пакет для створення UML-діаграм, відповідаючи найвищим стандартам програмної інженерії.

8.3. Microsoft Visio

Visio - рішення для побудови діаграм Microsoft. Згідно з твердженнями розробників, Visio дозволяє трансформувати складні технічні та бізнес-концепції у візуальну форму. Образотворчі можливості Visio є надзвичайно широкими. Використовуючи передвстановлені фігури Visio Professional, функціональність drag-and-drop і майстри, можна швидко та ефективно створювати зрозумілі та інформативні діаграми. Можливості Visio легко розширюються за допомогою нових шаблонів бізнес-діаграм, інтегруючи зовнішні джерела даних, сховища або колекції шаблонів, що зберігаються. Visio Professional тісно інтегрується з Microsoft Office Project, дозволяючи імпортувати завдання для членів команди.

З допомогою шаблонів UML можна створювати UML-діаграми статичної структури програмного забезпечення або проводити зворотне проектування за допомогою Visio 2003 Reverse Engineer Wizard.

8.4. StarUML, Dia, Draw.io

І нарешті, декілька чудових та абсолютно безкоштовних інструментів UML-моделювання. StarUML - це пакет з відкритим програмним кодом, написаний на Delphi, що працює під управлінням ОС сімейства Windows. StarUML підтримує UML 2.0 (з профілями) та MDA (Model Driven Architecture). Функціонал пакету можна розширювати за допомогою плагінів, що дозволяє кожному охочому створити свій власний модуль для StarUML на будь-якій COM-сумісній мові (C++, Delphi, C# тощо).

Dia - вільний кроссплатформенний редактор діаграм, частина GNOME Office, але може бути встановлений незалежно. Підтримує українську та російську мови. Dia може бути використаний для створення різноманітних видів діаграм: блок-схем алгоритмів програм, статичних структур UML, баз даних, діаграм сутність-зв'язок, радіоелектронних елементів, потокових діаграм, мережевих діаграм та інших.

Draw.io - інструмент для створення діаграм UML і блок-схем онлайн. Він нагадує MS Visio та, можливо, зроблений під нього, але в той час як додаток від Microsoft є платним, онлайн-сервіс Draw.io є безкоштовним. Сервіс підтримує декілька мов, включаючи російський інтерфейс. Сервіс дуже простий та зручний у використанні. За допомогою онлайн-сервісу Draw.io можна створювати діаграми, моделювання на UML, графіки, блок-схеми, форми, а також вставляти зображення в діаграми. Усі створені діаграми можна зберегти на комп'ютер, в Google Drive або Dropbox.

Rational Rose - це сімейство об’єктно-орієнтованих CASE-засобів від фірми Rational Software Corporation (у 2003 році її поглинула корпорація IBM), призначене для автоматизації процесів аналізу та проектування програмного забезпечення, а також для генерації коду на різних мовах і створення проектної документації. Rational Rose використовує метод об’єктно-орієнтованого аналізу та проектування, заснований на мові UML.

Поточна версія Rational Rose дозволяє генерувати коди програм для C++, Visual C++, Visual Basic, Java, PowerBuilder, CORBA Interface Definition Language (IDL), а також створювати описи баз даних для ANSI SQL, Oracle, MS SQL Server, IBM DB2 та Sybase. Вона також підтримує розробку проектної документації у вигляді діаграм та специфікацій. Крім того, Rational Rose включає засоби зворотного інжинірингу програм та баз даних.

Основою роботи Rational Rose є побудова UML-діаграм і специфікацій, що визначають архітектуру системи, її статичні та динамічні аспекти. У складі Rational Rose можна виділити шість основних структурних компонентів: репозиторій, графічний інтерфейс користувача, засоби перегляду проекту (браузер), засоби контролю проекту, засоби збору статистики та генератор

документів. Додатково до них додаються генератор коду (індивідуальний для кожної мови) та аналізатор для C++, що забезпечує зворотний інжиніринг.

Засоби автоматичної генерації коду на мові C++, використовуючи інформацію з UML-діаграм класів та компонентів, формують файли заголовків та файли описів класів і об'єктів. Створений таким чином скелет програми можна уточнити шляхом прямого програмування на мові C++.

Аналізатор коду C++ реалізований у вигляді окремого програмного модуля, призначеного для створення модулів проектів Rational Rose на основі інформації з початкових текстів на C++. Під час роботи аналізатор здійснює контроль правильності початкових текстів і діагностує помилки. Аналізатор має широкі можливості налаштування для ввода та виводу, що дозволяє визначити типи початкових файлів, базовий компілятор, інформацію для включення у модель і елементи для виводу на екран. Це забезпечує можливість повторного використання програмних компонентів у Rational Rose/C++.

У результаті розробки проекту за допомогою CASE-засобу Rational Rose формуються такі документи:

- UML-діаграми, що в сукупності є моделлю розроблюваної програмної системи;
- специфікації класів, об'єктів, атрибутів та операцій;
- шаблони програмних текстів.

Програмні тексти виступають як заготовки для подальшої роботи програмістів. Склад інформації, яка включається в програмні файли, визначається або за замовчуванням, або на розсуд користувача.

Для підтримки командної роботи над проектом на кожному етапі життєвого циклу програмного забезпечення (ПЗ) використовується інтегрований набір продуктів Rational Suite. Вони існують у наступних варіантах:

- Rational Suite AnalystStudio - призначений для визначення та управління повним набором вимог до системи, яка розробляється;
- Rational Suite DevelopmentStudio - використовується для проектування та реалізації ПЗ;
- Rational Suite TestStudio - включає в себе набір продуктів для автоматизованого тестування додатків;
- Rational Suite Enterprise - забезпечує підтримку всього життєвого циклу ПЗ і призначений як для менеджерів проекту, так і для окремих розробників, що виконують кілька функціональних ролей.

Інтерфейс Rational Rose складається з п'яти основних елементів: браузера, вікна документації, панелей інструментів, вікна діаграм та журналу (log). Їх призначення є наступним:

- браузер (browser) - використовується для швидкої навігації по моделі;
- вікно документації (documentation window) - застосовується для роботи з текстовим описом елементів моделі;
- панелі інструментів (toolbars) - забезпечують швидкий доступ до

найпоширеніших команд;

- вікно діаграм (diagram window) - використовується для перегляда та редагування однієї або декількох діаграм UML;
- журнал (log) - застосовується для перегляда помилок та звітів про результати виконання різних команд.

На рисунку 8.1 показані різні частини інтерфейсу Rational Rose.

Браузер - це ієрархічна структура, що дозволяє здійснювати навігацію по моделі. Усе, що додається в модель - дійові особи, варіанти використання, класи, компоненти - буде показано у вікні браузера. За допомогою браузера можна:

- додавати в модель елементи (дійові особи, варіанти використання, класи, компоненти, діаграми і так далі);
- переглядати існуючі елементи та зв’язки між елементами моделі;
- переміщати та перейменовувати елементи моделі;
- додавати елементи моделі до діаграми.
- групувати елементи в пакети

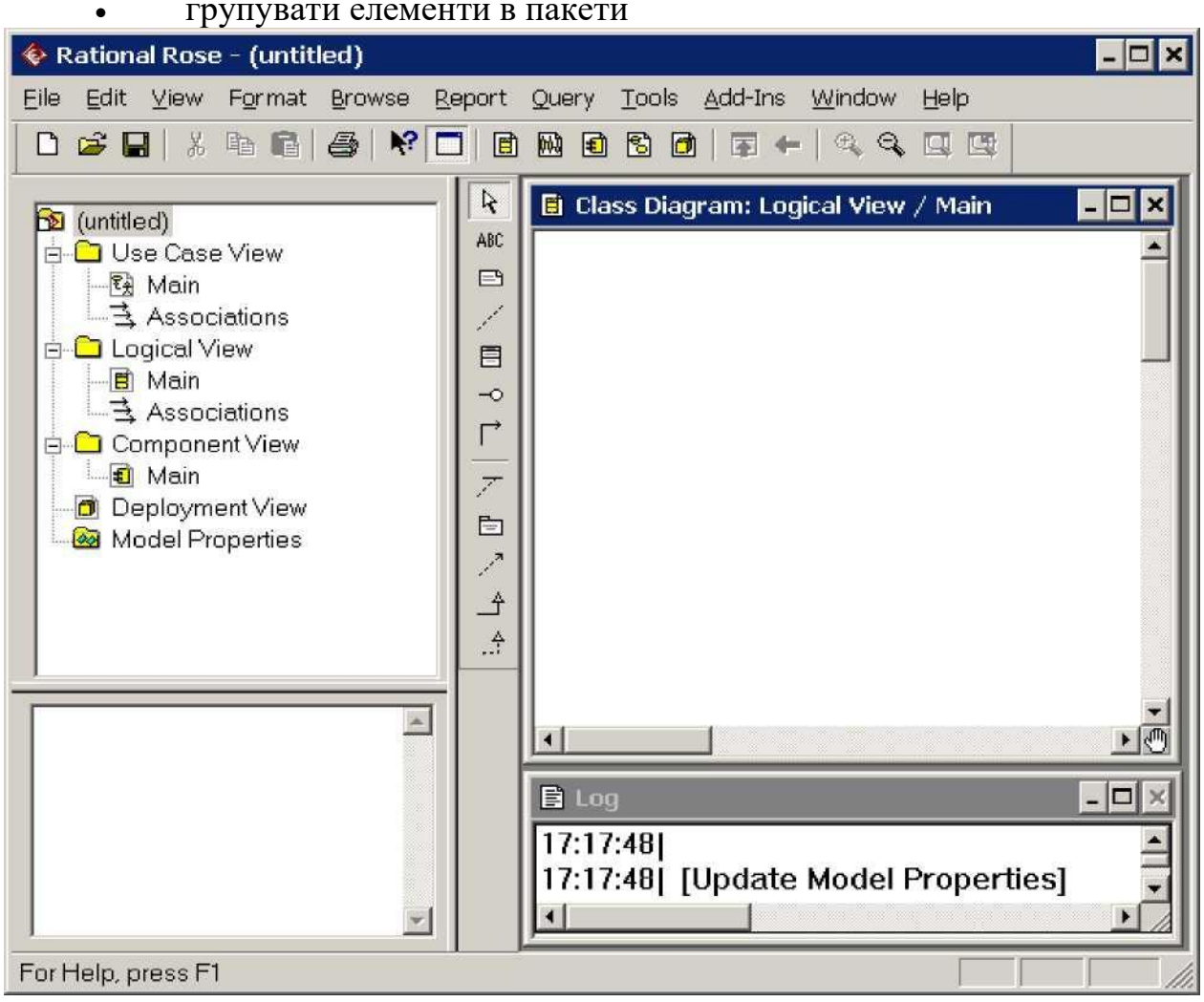

Рис. 8.1 - Інтерфейс Rational Rose

Браузер підтримує чотири види представлень: представлення варіантів використання, компонентів, розміщення та логічне представлення. Усі вони

разом з елементами моделі, що в них містяться, детально описані нижче.

Структура браузера побудована у вигляді дерева. Кожен елемент моделі може містити підпорядковані елементи, що розташовані нижче у ієрархії. Знак «-» біля елементу вказує на те, що його гілка повністю розгорнута, тоді як знак «+» свідчить про те, що його гілка згорнута.

Вікно документації дозволяє створювати опис елементів моделі Rose. Наприклад, можна додати короткий опис до кожного учасника. При документуванні класу все, що буде введено у вікно документації, пізніше з’явиться у вигляді коментаря в згенерованому коді, що позбавляє необхідності вручну додавати ці коментарі. Також, ця документація буде включена до звітів, що створюються в середовищі Rose.

Панелі інструментів забезпечують швидкий доступ до найчастіше використовуваних команд. В середовищі Rose існує два типи панелей інструментів: стандартна панель і панель діаграм. Стандартна панель завжди видима, її кнопки відповідають командам, які можуть бути використані для роботи з будь-якою діаграмою. Панель діаграм індивідуальна для кожного типу діаграм UML.

Усі панелі інструментів можуть бути налаштовані користувачем. Для цього потрібно обрати пункт меню Tools > Options, а потім перейти на вкладку Toolbars.

Щоб показати або приховати стандартну панель інструментів чи панель інструментів діаграм:

1. Виберіть пункт Tools > Options.
2. Перейдіть на вкладку Toolbars.
3. Щоб змінити видимість стандартної панелі інструментів, поставте або зніміть позначку з контрольного перемикача Show Standard ToolBar (або Show Diagram ToolBar).

Збільшення розміру кнопок на панелі інструментів:

1. Натисніть правою кнопкою миші на потрібній панелі інструментів.
2. У спливаючому меню оберіть пункт Use Large Buttons (Використати великі кнопки).

Налаштування панелі інструментів:

1. Натисніть правою кнопкою миші на відповідній панелі інструментів.
2. У меню, що з'явилося, виберіть пункт Customize (Налаштувати).
3. Для додавання або видалення кнопок, оберіть потрібну кнопку і натисніть кнопку Add (Додати) або Remove (Видалити).
4. Вікно діаграми.

У вікні діаграми відображаються одна або декілька діаграм UML моделі. При внесенні змін в елементи діаграми, Rose автоматично оновлює браузер. Так само, при зміні елементів за допомогою браузера, Rose автоматично оновлює відповідні діаграми. Це допомагає підтримувати модель у консистентному стані.

Під час роботи над вашою моделлю, певна інформація буде зберігатися у вікні журналу. Наприклад, туди потрапляють повідомлення про помилки, що виникають під час генерації коду. Журнал не можна повністю закрити, але його вікно можна мінімізувати.

У моделі Rose підтримуються чотири види представлення (views) - представлення варіантів використання, логічне представлення, представлення компонентів і представлення розміщення. Кожне з цих представлень має свої цілі та аудиторію. У подальших підрозділах цього розділу ми коротко розглянемо кожне з цих представлень, а в решті частини книги детально обговоримо елементи моделі, що містяться в них.

Представлення варіантів використання. Це представлення містить усіх дійових осіб, усі варіанти використання та їх діаграми для конкретної системи. Воно може також включати деякі діаграми послідовності та кооперативні діаграми. На рис. 8.2 показано, як виглядає представлення варіантів використання у браузері Rose.

Представлення варіантів використання містить: дійових осіб; варіанти використання; документацію по варіантах використання, яка деталізує процеси (потоки подій), що відбуваються у варіантах використання, включаючи обробку помилок.

Піктограмами зображені зовнішні файли, прикріплені до моделі Rose. Вид піктограми залежить від додатка, що використовується для документування потоку подій. У даному випадку (рис. 8.3) використовувався Microsoft Word.

1. Діаграми варіантів використання. Зазвичай у системі є декілька таких діаграм, кожна з яких відображає підмножину дійових осіб і/або варіантів використання.
2. Пакети, що являють собою сукупності різноманітних сценаріїв використання та/або активних суб'єктів.

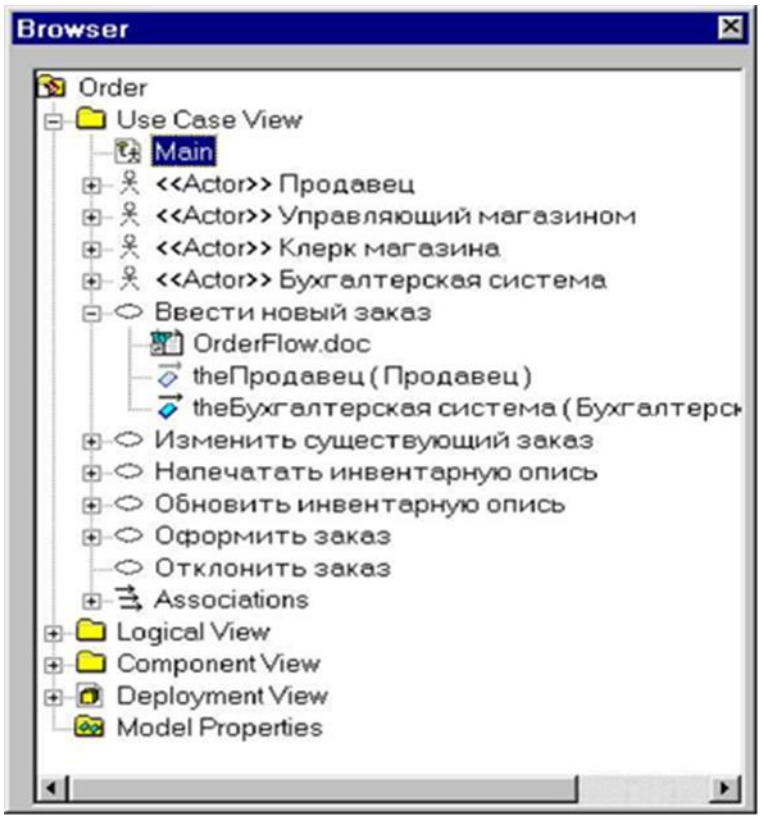

Рис. 8.2 - Репрезентація варіантів використання.

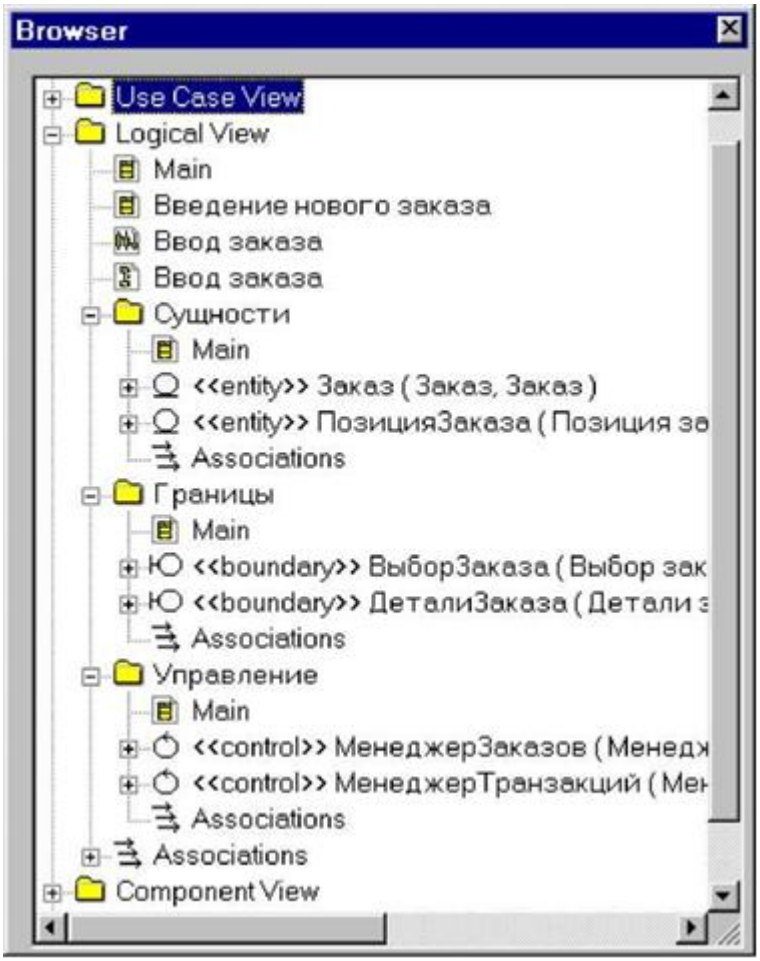

Рис. 8.3 - Логічне представлення системи

Логічне представлення, відображене на рис. 8.3, зосереджується на методах реалізації системою поведінки, описаної у варіантах використання. Воно надає детальну картину складових елементів системи і описує взаємодію між цими елементами. Логічне представлення включає, серед іншого, конкретні необхідні класи, діаграми класів і діаграми станів, за допомогою яких створюється детальний проект системи, що розробляється.

Логічне представлення містить наступні компоненти:

1. Класи.
2. Діаграми класів. Зазвичай для опису системи використовується декілька діаграм класів, кожна з яких відображає певну підмножину всіх класів системи.
3. Діаграми взаємодії, які застосовуються для відображення об’єктів, що беруть участь в одному потоці подій варіанту використання.
4. Діаграми станів.
5. Пакети, що представляють собою групи взаємопов'язаних класів.

Представлення компонентів включає кілька ключових елементів:

1. **Компоненти** - фізичні модулі коду, що складаються з окремих частин програмного забезпечення.
2. **Діаграми Компонентів** - графічні зображення, що відображають взаємозв'язки між компонентами.
3. **Пакети** - групи пов'язаних між собою компонентів, що утворюють логічну структуру.

Останнє представлення в Rose - це представлення розміщення, яке описує фізичну організацію системи. Це представлення може істотно відрізнятися від її логічної архітектури і включає:

1. **Процеси** - потоки (threads), що виконуються в спеціально відведеній для них області пам'яті.
2. **Процесори** - обчислювальні пристрої, здатні обробляти дані. Кожен процес виконується на одному або декількох процесорах.
3. **Пристрої** - апаратні засоби, що не здатні обробляти дані, такі як термінали введення-виведення і принтери.
4. **Діаграма Розміщення** - графічне відображення фізичного розташування процесів, процесорів і пристроїв.

Параметри Налаштування Відображення

Зображення атрибутів і операцій на діаграмах класів у Rose можна налаштовувати за допомогою наступних параметрів:

1. Показ усіх атрибутів і операцій.
2. Приховування операцій.
3. Приховування атрибутів.
4. Показ лише деяких атрибутів або операцій.
5. Показ операцій з повними сигнатурами або тільки їх іменами.
6. Показ або приховування видимості атрибутів і операцій.
7. Показ або приховування стереотипів атрибутів і операцій.

Значення цих параметрів за замовчуванням можна встановити через

меню Tools > Options.

Наприклад, для показу всіх атрибутів класу потрібно:

1. Виділити потрібний клас на діаграмі.
2. Клацнути правою кнопкою миші, щоб відкрити контекстне меню.
3. Вибрати Options > Show All Attributes.

Для налаштування відображення лише вибраних атрибутів:

1. Виділити потрібний клас на діаграмі.
2. Клацнути правою кнопкою миші для відкриття контекстного меню.
3. Вибрати Options > Select Compartment Items.
4. Вказати необхідні атрибути у вікні Edit Compartment.

Для приховування всіх атрибутів класу:

1. Виділити потрібний клас на діаграмі.
2. Клацнути правою кнопкою миші для відкриття контекстного меню.
3. Вибрати Options > Suppress Attributes.

Змінити параметри відображення атрибутів за замовчуванням можна наступним чином:

1. Вибрати в меню моделі пункт Tools > Options.
2. Перейти на вкладку Diagram.
3. Використати контрольні перемикачі Suppress Attributes і Show All Attributes для встановлення потрібних значень за замовчуванням. Ці зміни вплинуть тільки на нові діаграми, а існуючі діаграми класів залишаться без змін.

Аналогічно до атрибутів, існують різні варіанти представлення операцій на діаграмах, які можна налаштовувати за допомогою відповідних параметрів відображення. А саме, продемонструвати всі операції, відобразити лише окремі операції, сховати всі операції, призупитинити виведення операцій.

Крім того, можна: показати лише ім'я операції (на діаграмі буде представлено тільки ім'я операції без аргументів або типу значення, що повертається), показати повну сигнатуру операції (на діаграмі буде представлено не лише ім'я операції, але і всі її параметри, типи даних параметрів і тип значення операції, що повертається).

Для відображення всіх операцій класу:

- виберіть на діаграмі потрібний клас;
- клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню;
- виберіть у ньому Options > Show All Operations.

Для відображення тільки вибраних операцій класу:

- виберіть на діаграмі потрібний клас;
- клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню;
- виберіть у ньому Options > Select Compartment Items;
- вкажіть потрібні операції у вікні Edit Compartment.

Для подавлення виведення всіх операцій класу на діаграмі:

• виберіть на діаграмі потрібний клас;

• клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню;

• виберіть у ньому Options > Suppress Operations.

Для відображення сигнатури операції на діаграмі класів:

• виберіть на діаграмі потрібний клас;

• клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню

• виберіть у ньому Options > Show Operation Signature.

Для зміни за замовчуванням виду операції:

• у меню моделі виберіть пункт Tools > Options;

• перейдіть на вкладку Diagram;

• скористайтеся контрольними перемикачами Suppress Operations, Show All Operations і Show Operation Signatures для установки параметрів відображення операцій за замовчуванням.

Для відображення видимості атрибуту або операції класу:

- виберіть на діаграмі потрібний клас.
- клацніть на ньому правою кнопкою миші, щоб відкрити контекстно-залежне меню.
- виберіть у ньому Options > Show Visibility.

Для зміни параметра показу видимості за замовчуванням:

- у меню моделі виберіть пункт Tools > Options;
- перейдіть на вкладку Diagram;
- скористайтеся контрольним перемикачем Show Visibility для установки параметрів відображення видимості за замовчуванням.

Для перемикання між нотаціями видимості Rose і UML:

- у меню моделі виберіть пункт Tools > Options;
- перейдіть на вкладку Notation;
- скористайтеся перемикачем Visibility as Icons для перемикання між нотаціями. Якщо цей перемикач увімкнено, використовуватиметься нотація Rose. Якщо ні - нотація UML. Зміна цього параметра вплине лише на нові діаграми. Існуючі діаграми класів залишаться без змін.

Контрольні питання до розділу 8

1. Призначення CASE-засобу *IBM Rational Rose.*
2. Які версії продукту *Rational Rose* ви знаєте?
3. Структура і функції *Rational Rose.*
4. Які п’ять основних елементів інтерфейсу *Rose* ви знаєте?
5. Призначення панелі інструментів *Rose.*
6. Назвіть основне призначення *Браузера* в *Rose*

РОЗДІЛ 9. ЗАГАЛЬНІ ШЛЯХИ ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ КОРИСТУВАЧА

9.1. Правила та принципи розробки інтерфейсу користувача

Експерти зазвичай формулюють набір принципів і правил, що дозволяють оцінювати зручність інтерфейсу та пропонувати рішення для її підвищення. Ось ці правила:

Правило доступності. Система повинна бути настільки зрозумілою, щоб користувач, який ніколи раніше її не бачив, але добре орієнтується в предметній області, міг без жодного навчання почати її використовувати. Це правило слугує певним ідеалом, до якого слід прагнути, оскільки на практиці досягти такої доступності майже ніколи не вдається. Проте фахівці постійно працюють над цим.

Правило ефективності. Система не повинна заважати ефективній роботі досвідчених користувачів, які працюють з нею протягом тривалого часу. Очевидним прикладом порушення цього правила є орієнтація системи тільки на новачків, використання засобів, які обмежують можливості експертів, але допомагають недосвідченим користувачам уникати помилок.

Правило безперервного розвитку. Система повинна сприяти постійному зростанню знань, умінь і навичок користувачів та адаптуватися до їхнього досвіду, що змінюється. Поганим прикладом є надання тільки базових можливостей або залишення початківців з складним інтерфейсом, яким впевнено користуються лише експерти. Порушення безперервності при переході від одного набору можливостей до іншого також створює незручності, оскільки користувач змушений адаптуватися до нових умов.

Користувачів можна розділити на три групи: новачки, досвідчені і середні. Новачкам потрібна допомога в освоєнні нової системи і контроль за їхніми діями, тоді як досвідченим користувачам потрібні висока ефективність і гнучкість. Про середніх користувачів часто забувають, хоча більшість користувачів програмного забезпечення належать саме до цієї категорії. Їм необхідні ефективність і можливість швидко отримувати допомогу з різних питань.

Правило дотримання контексту. Система повинна бути адаптована до конкретного контексту, в якому вона буде використовуватися. Це означає, що система повинна функціонувати не "взагалі", а в конкретному оточенні, де будуть враховані специфіка і обсяг вхідних та вихідних даних, тип і цілі

Наведені вище правила визначають загальні вимоги до зручного інтерфейсу. Наступні принципи допомагають знаходити рішення для підвищення зручності користувацького інтерфейсу:

Принцип структуризації. Інтерфейс користувача повинен бути логічно структурованим. Елементи, близькі за змістом, повинні бути зв'язані між собою видимим чином, тоді як незалежні – розділені; схожі елементи повинні виглядати схоже, а несхожі – відрізнятися.

Принцип простоти. Найбільш поширені операції повинні виконуватися

максимально просто. При цьому мають бути видимі посилання на складніші процедури.

Принцип видимості. Усі функції і дані, необхідні для виконання певного завдання, повинні бути видні, коли користувач намагається його вирішити.

Принцип зворотного зв'язку. Користувач повинен отримувати повідомлення про дії системи та важливі події всередині неї. Повідомлення мають бути інформативними, короткими, однозначними і зрозумілими користувачеві.

Принцип толерантності. Інтерфейс повинен бути гнучким і терпимим до помилок користувача. Збитки від помилок слід знижувати за рахунок можливості відміни і повтору дій та розумної інтерпретації будь-яких розумних дій і введених даних користувача. Варто уникати взаємодії на основі обмеження свободи користувача.

Принцип повторного використання. Необхідно намагатися багаторазово використовувати внутрішні та зовнішні компоненти, забезпечуючи уніфікованість інтерфейсу та схожість між його елементами.

9.2. Взаємодія між користувачем і комп’ютером

Людиномашинний інтерфейс забезпечує зв'язок між користувачем і комп'ютером, дозволяючи досягати поставлених цілей і успішно вирішувати завдання. Взаємодія – це обмін діями та реакціями на ці дії між комп'ютером і користувачем. Існує кілька стилів взаємодії, які поділяються на два види [38]:

1. Використання командного інтерфейсу – введення команд текстовими засобами.
2. Безпосереднє маніпулювання.

Таким чином, існують різні способи взаємодії користувача з комп'ютером:

- командні мови – користувач керує системою, вводячи відповідні команди в текстовому режимі;
- питання і відповідь – діалог, де комп'ютер ставить питання, а користувач відповідає (або навпаки);
- форми – користувач заповнює форми або поля діалогу, вводячи дані у відповідні поля;
- меню – користувач обирає необхідні пункти з ряду опцій для керування системою;
- пряме маніпулювання – користувач керує об'єктами на екрані за допомогою маніпулятора, наприклад, миші.

У програмній системі, що розробляється, використовується комплексний підхід до створення інтерфейсу, що включає пряме маніпулювання, меню, форми і діалоги.

Мета створення ергономічного інтерфейсу полягає в тому, щоб відображати інформацію настільки ефективно, наскільки це можливо для

людського сприйняття, і структурувати відображення на дисплеї так, щоб привертати увагу до найважливіших елементів. Основна мета – мінімізувати загальну кількість інформації на екрані і представляти тільки те, що є необхідним для користувача.

9.3. Розміщення інформації на екрані

Кількість інформації, відображеної на екрані, називається екранною щільністю. Дослідження показують, що чим менша екранна щільність, тим доступнішою та зрозумілішою є інформація для користувача. Навпаки, велика екранна щільність може ускладнити засвоєння та розуміння інформації. Проте досвідчені користувачі часто віддають перевагу інтерфейсам з високою екранною щільністю. Інформацію на екрані можна групувати і впорядковувати у значимі частини, використовуючи кадри, колірне кодування, рамки, негативне зображення або інші методи для привернення уваги.

Виділення елементів інтерфейсу яскравістю. Щоб привернути увагу до певних елементів інтерфейсу, їх можна зробити яскравішими на тлі інших, темніших елементів. Однак важливо не перестаратися з цим методом, оскільки велика кількість яскравих елементів може викликати дискомфорт у користувача та призвести до перевантаження інтерфейсу. Цей метод слід використовувати лише за необхідності. Існує кілька способів виділення яскравістю:

- рух (миготіння або зміна позиції) – ефективний метод;
- яскравість – не дуже ефективний метод, оскільки люди здатні розрізнити лише декілька рівнів яскравості;
- колір – може бути надзвичайно ефективним;
- форма (символ, шрифт, форма символу) – використовується для диференціації різних категорій даних;
- розмір (тексту, символів) – зазвичай збільшення виділеного об'єкта в 1,5 рази;
- відтінення (різна текстура об'єктів) – ефективний метод для привернення уваги до певної частини екрану;
- оточення (підкреслення, рамки, інвертоване зображення) – дуже ефективний спосіб.

Використання кольору при проектуванні ергономічного інтерфейсу. Колір може покращити інтерфейс, але для багатьох систем використання кольору практично не впливає на ефективність роботи користувача. Основне призначення кольору – створення більш цікавих для користувачів інтерфейсів. Проте в деяких випадках колір може допомогти при групуванні інформації, виділенні відмінностей між даними або при передачі простих повідомлень (помилки, стани тощо).

Колір – потужний візуальний інструмент, і застосовувати його потрібно дуже обережно, щоб уникнути дискомфорту від неправильних колірних

комбінацій. Деякі принципи використання кольору при проектуванні ергономічного інтерфейсу:

- обмежити кількість кольорів до 4 на одному екрані та до 7 для послідовності екранів;
- для неактивних елементів використовувати бліді кольори;
- при відображенні стану зазвичай червоний означає небезпеку (стоп), зелений – продовження роботи, жовтий – попередження;
- для привернення уваги найефективніші білі, жовті та червоні кольори;
- для впорядкування або розділення даних можна використовувати спектр із 7 кольорів (веселка);
- для угрупування даних краще використовувати сусідні кольори спектру: помаранчево-жовті, синьо-фіолетові.

Важливо зазначити, що близько 9% людей не розрізняють кольорів (зазвичай червоно-зелені поєднання); проте ці люди можуть розрізняти чорнобілі відтінки. Тому проектувальники автоматизованих систем повинні враховувати цей факт, перевіряючи, чи не порушує використання кольорів сприйняття користувачів цієї категорії.

Несуперечність і стандартизація. Дані на екрані слід розташовувати так, щоб користувач знав, де знайти та де очікувати виведення необхідної інформації. Так, інформація, яка потребує негайної уваги, повинна завжди відображатися на видному місці, щоб захопити увагу користувача (наприклад, попереджувальні повідомлення і повідомлення про помилки). Інформація, яка рідко потрібна (наприклад, довідка), не повинна відображатися постійно, але має бути доступною за потреби. Наприклад, іконка "довідка" або відповідна опція меню має бути доступна на кожному екрані.

Тексти і діалоги. Деякі принципи, які необхідно враховувати при створенні текстових діалогів і відображень:

- текст у нижньому регістрі читається приблизно на 13% швидше, ніж текст у верхньому регістрі;
- символи верхнього регістру ефективні для передачі інформації, яка повинна привернути увагу. не використовуйте верхній регістр, якщо ви не хочете виділяти інформацію;
- вирівняний по правому краю текст важче читати, ніж рівномірно розподілений текст з невирівняним правим полем;
- оптимальний інтервал між рядками дорівнює або трохи більше за висоту символів.

Меню – необхідний елемент автоматизованої системи, який дозволяє користувачеві виконувати завдання всередині додатка і керувати процесом рішення. Меню – це набір опцій, відображених на екрані, де користувачі можуть вибирати і виконувати дії, змінюючи таким чином стан інтерфейсу. Перевага меню в тому, що користувачі не повинні пам'ятати назву елемента або дії, яку вони хочуть виконати; вони мають лише розпізнати її серед пунктів меню.

Меню можуть використовувати навіть недосвідчені користувачі, проте

проект меню повинен бути ретельно продуманий. Щоб меню було ефективним, назви його пунктів повинні бути очевидними. Меню може займати багато екранного простору, але вирішенням цієї проблеми може бути використання спливаючого або спадаючого меню, яке викликається клацанням по піктограмі, рядку меню або іншому об'єкту.

Під час проектування системи меню додатка важливо обрати найкращий спосіб відображення меню, щоб воно було зрозумілим та зручним у використанні. Зазвичай, команди меню впорядковуються ієрархічним чином. Основна задача полягає у правильному розподілі різних пунктів меню по рівнях та їх згрупуванні.

Принципи проектування меню:

1. Відповідність структурі завдання. Структура меню повинна відповідати структурі завдання, яке розв'язується системою. Організація меню має відображати найефективнішу послідовність кроків для вирішення задачі.
2. Чіткість та лаконічність. Пункти меню повинні бути короткими, граматично правильними та відповідати своєму заголовку. Порядок пунктів меню вибирається відповідно до угоди, частоти і порядку використання, а також залежно від потреб завдання або користувача.
3. Гнучкість у виборі. Вибір пунктів меню має бути можливим декількома способами – за допомогою клавіатури, миші та інших елементів інтерфейсу. Важливо зафіксувати комбінації клавіш, що легко запам'ятовуються, для швидшого доступу до пунктів меню, що значно економить час.

Форми – основний елемент інтерфейсу, призначений для зручного введення і перегляду даних, стану та повідомлень автоматизованої системи.

Основні принципи проектування форм:

1. Зручність та зрозумілість. Форма повинна бути розроблена так, щоб забезпечувати зручне, зрозуміле і швидке вирішення поставленої задачі. Якщо форма переноситься з паперової форми, пересування по суміжних полях не повинне викликати труднощів у користувача.
2. Логічне розміщення інформації. Розташування інформаційних одиниць на формі повинно відповідати логіці її майбутнього використання. Це залежить від необхідної послідовності доступу до інформаційних одиниць, частоти їх використання та відносної важливості елементів.
3. Відділення логічних груп. Логічні групи елементів необхідно відділяти пропусками, рядками, кольоровими або іншими візуальними засобами.
4. Відображення взаємозалежних елементів. Взаємозалежні або пов'язані елементи повинні відображатися в одній формі.

Під час розробки форм необхідно продумати і вказати, які кнопки у смузі системного меню мають бути доступні в конкретному вікні, чи повинне вікно дозволяти користувачеві змінювати його розмір, яким має бути заголовок вікна. Без особливої необхідності не слід робити вікна зі змінюваними розмірами. Якщо ж вікно має змінюваний розмір, необхідно забезпечити, щоб

компоненти у вікні також змінювали свої розміри або розташування, рівномірно розподіляючись по площі вікна, і не залишали порожніх місць.

При проектуванні форм важливо обмежити кількість кольорів та приділяти увагу їх правильному поєднанню. Для фону форми обираються нейтральні кольори (світло-сірі). Колір не повинен використовуватися як основний засіб передачі інформації, натомість потрібно вибирати системні кольори, які користувач може налаштовувати на свій розсуд. Елементи управління та функціонально пов'язані з ними компоненти екрану слід зорово об'єднувати в групи, заголовки яких коротко і чітко пояснюють їх призначення.

Кожне вікно повинне мати центральну тему, яка підпорядковується його композиції. Користувач повинен розуміти, для чого призначене це вікно і що в ньому найважливіше. Не можна перевантажувати вікно великою кількістю елементів управління введення і відображення інформації. Також неприпустимо, щоб схожі за функціями елементи управління в різних вікнах називалися по-різному або розташовувалися в різних місцях вікон.

Важливо також врахувати, як додаток вписується в загальну організацію робочого простору системи та як він взаємодіє з іншими програмами. Для окремих полів заголовок має бути вирівняний по лівому краю; для полів списків заголовок має бути вищий і лівіший щодо основного поля; числові поля вирівнюються по правому краю. Довгі колонкові поля або довгі стовпці інформаційних одиниць слід об'єднувати в групи по п'ять елементів, розділяючи їх порожнім рядком – це допомагає користувачеві обробляти інформацію по виділених групах.

У формах з великою кількістю інформації важливо використовувати назви розділів, які однозначно свідчать про характер інформації. Необхідно чітко розділяти відображення заголовків і полів введення. Заголовки мають бути короткими, знайомими і змістовними. Поля, що необов'язкові для заповнення або не мають особливої важливості, повинні візуально відрізнятися (кольором або іншими ефектами) від важливих і обов'язкових для заповнення полів.

Слід забезпечити введення значень за замовчуванням у всі поля, де це можливо і не дратує користувача. Можна призначати клавіші або коди для введення часто використовуваних значень. Вхідні дані мають бути значущими і загальноприйнятими. Не слід об'єднувати поля введення чисел і символів, оскільки числові і алфавітні клавіші розташовані незручно відносно один одного на клавіатурі. Також потрібно уникати частого перемикання між верхнім і нижнім регістрами для прискорення введення даних. Не можна вимагати від користувача введення незначних цифр (наприклад, замість 00000010 користувач повинен вводити лише 10). Аналогічно, не можна вимагати введення інформації, яка вже була введена або може бути автоматично отримана з системи. Бажано використовувати значення за замовчуванням, щоб мінімізувати процес введення інформації.

Навігація забезпечує можливість переміщення між різними екранами,

інформаційними одиницями і підпрограмами в автоматизованій системі. У повноцінній системі користувач завжди може отримати інформацію про стан системи, процес виконання або активну підпрограму.

Загальні принципи проектування:

1. Використання заголовків сторінок для кожного екрану, номерів сторінок, рядків і стовпців, відображення поточного імені файлу вгорі екрану.
2. Тип системи навігації залежить від прийнятого стилю інтерфейсу. Для інтерфейсів мови команд існує дуже мало способів забезпечення повноцінної навігації. У інтерфейсах з меню можна використовувати ієрархічно структуроване меню. Діалогові інтерфейси самі по собі захищають користувача від помилкових дій. Інформація про стан зазвичай відображається внизу екрану і містить дані про кількість записів, число оброблених одиниць, процес друку, черги друку тощо.

Проектування повідомлень відіграє ключову роль у спрямуванні дій користувача у правильному напрямку, наданні підказок та попереджень під час виконання необхідних кроків для розв'язання завдань. Повідомлення також містять підтвердження дій з боку користувача та системи, вказуючи на успішне завершення завдання або на причини його невдачі. Повідомлення можуть бути представлені у вигляді діалогів, екранних заставок тощо.

Вони можуть запропонувати користувачу:

- вибрати опцію або набір опцій з запропонованих альтернатив;
- ввести необхідну інформацію;
- вибрати опцію з набору, який може змінюватися в залежності від поточного контексту;
- підтвердити введену інформацію перед продовженням роботи.

Повідомлення можуть бути розміщені в модальних діалогових вікнах, які змушують користувача відповісти на питання перед виконанням будь-яких інших дій. Це може бути корисно, коли система потребує обмірковування користувачем свого рішення перед продовженням роботи. Немодальні діалогові вікна дозволяють взаємодіяти з іншими елементами інтерфейсу, тоді як саме вікно може бути проігнороване.

Звичайна людина у нормальному стані часто здійснює різні помилки. На відміну від комп’ютера, людина є адаптивною аналоговою системою, і успішність її дій визначається не стільки точністю виконання, скільки здатністю швидко досягати наближеного до потрібного результату і виправлятися у разі потреби.

Помилки користувача можуть виникати через неправильне розуміння дії або її послідовності, або бути випадковими, наприклад, друкарські помилки. Випадкові помилки можна класифікувати на шість підвидів:

- неправильний вибір опції (наприклад, випадкове натискання кнопки "Вихід");
- втрата активності, коли користувач забуває послідовність дій;
- помилка режиму або стану, коли користувач вважає, що знаходиться в одному стані, а насправді - в іншому (наприклад, режим вставки

замість режиму друку поверх тексту).

У будь-якій програмній системі користувач буде допускати помилки, тому необхідно передбачити захист від них. Техніки захисту включають:

- примусові дії, що запобігають або ускладнюють виникнення помилок;
- інформативні повідомлення про помилки;
- забезпечення нормальної діагностики, яка пояснює суть помилки і вказує шляхи її виправлення.

Основні принципи обробки помилок у формах введення включають:

- можливість посимвольного редагування введених даних для виправлення помилок;
- повернення курсору в поле з помилковими даними і виділення цього поля;
- виведення значущих повідомлень про помилки, використовуючи стиль мови користувача;
- повідомлення, що пояснюють суть помилки і пропонують шляхи її усунення.

Ефективність запобігання та подолання помилок тим вища, чим рідше користувачі помиляються при роботі з інтерфейсом і чим менше часу та зусиль потрібно для виправлення наслідків вже допущених помилок.

9.4. Вибір стратегії тестування і розробка тестів

Тестування визначається як процес виконання програми з метою виявлення помилок, тоді як відладка полягає в локалізації та виправленні цих помилок. Існують різні рівні тестування, кожен з яких має свої особливості та призначення.

Модульне тестування - це процес перевірки окремих програмних процедур (модулів) і підпрограм, що складають частини більш складних програмних систем. Цей вид тестування зазвичай виконується безпосереднім розробником і дозволяє детально аналізувати внутрішні структури та потоки даних в кожному модулі.

Інтеграційне тестування - це перевірка взаємодії між окремими модулями, що передує тестуванню всієї системи як єдиного цілого. В ході цього тестування перевіряються зв’язки між модулями, їх сумісність і функціональність. Інтеграційне тестування здійснюється незалежним тестувальником і є частиною загального процесу тестування. Елементи інтеграційного тестування включають:

- перевірка функціональності – верифікація відповідності функцій, виконуваних сукупностями модулів, вимогам специфікації;
- перевірка проміжних результатів – контроль наявності та коректності всіх проміжних результатів і файлів;
- перевірка інтеграції – верифікація коректності передачі інформації між модулями.

Системне тестування проводиться з метою перевірки програмної

системи в цілому на відповідність вимогам специфікацій замовника. Це тестування виконується незалежним тестувальником після успішного завершення інтеграційного тестування. Елементи системного тестування включають:

- граничне тестування – перевірка в граничних умовах;
- прогоночне тестування – верифікація всіх функціональних характеристик системи;
- цільове тестування – тестування на цільовій платформі;
- перевірка документації – контроль коректності призначеної для користувача документації;
- інші тести – визначаються тестувальником залежно від специфіки системи.

Вихідне тестування – завершальний етап тестування, що перевіряє готовність програмного продукту для постачання замовнику. Цей вид тестування також здійснюється незалежним тестувальником.

Приймальне тестування проводиться організацією, відповідальною за інсталяцію, супровід програмної системи та навчання кінцевих користувачів.

Технологія тестування, яка застосовується на етапі розробки програмного забезпечення, відома як тестування «скляного ящика» («білого ящика»), що протиставляється класичному поняттю тестування «чорного ящика».

Тестування «чорного ящика» - це метод, при якому внутрішня структура програми залишається невідомою для тестувальника. Тестувальник вводить дані та аналізує результат, не маючи уявлення про те, як саме працює програма.

Тестування «скляного ящика» - це метод, при якому тестувальник, який зазвичай є програмістом, розробляє тести на основі знань про початковий код програми, до якого він має повний доступ. Цей метод має кілька переваг:

1. Спрямованість тестування. Програміст може тестувати програму частинами, розробляючи спеціальні тестові підпрограми, які викликають тестований модуль і передають йому необхідні дані. Окремий модуль легше тестувати саме як «скляний ящик».
2. Повне охоплення коду. Програміст може визначити, які саме фрагменти коду працюють у кожному тесті. Він бачить, які гілки коду залишилися непротестованими, і може підібрати умови, за яких вони будуть протестовані.
3. Управління потоком команд. Програміст знає, яка функція повинна виконуватися в програмі наступною і яким має бути її поточний стан. Щоб з’ясувати, чи працює програма правильно, програміст може додати налагоджувальні команди або скористатися спеціальним програмним засобом – відладчиком.
4. Відстежування цілісності даних. Програміст знає, яка частина програми повинна змінювати кожен елемент даних. Відстежуючи стан даних, він може виявити помилки, такі як неправильна зміна даних або невірна їх

інтерпретація.

Тестування «скляного ящика» є невід'ємною частиною процесу програмування. Програмісти тестують кожен модуль після його написання і повторно після інтеграції в систему. Модульне тестування може здійснюватися за технологіями структурного або функціонального тестування, або обох одночасно.

Усі програмні помилки можуть бути класифіковані за відповідними категоріями. Розглянемо ті, що зустрічаються найчастіше:

1. Функціональні недоліки виникають, коли програма не виконує своїх функцій належним чином, реалізує їх частково або взагалі не виконує. Функціональні характеристики програми мають бути детально описані у специфікації, на основі якої тестувальник будує свою роботу.
2. Недоліки призначеного для користувача інтерфейсу визначаються зручністю та коректністю роботи інтерфейсу. Оцінка інтерфейсу можлива тільки під час його використання, і бажано, щоб у цьому процесі брав участь кінцевий користувач.
3. Недостатня продуктивність характеризується низькою швидкістю роботи програми, що може бути критичним параметром, зазначеним у вимогах замовника. Якщо у користувача виникає відчуття повільної роботи програми, це свідчить про недоліки продуктивності.
4. Некоректна обробка помилок. Процедури обробки помилок є важливою частиною програми. Програма повинна коректно визначати помилки і видавати відповідні повідомлення.
5. Некоректна обробка граничних умов. Граничні умови виникають у випадках, коли застосовуються поняття "більше" або "менше", "раніше" або "пізніше", "коротше" або "довше". Програма має бути перевірена на межах діапазону для будь-якого аспекту її роботи.
6. Помилки обчислень включають помилки, що виникають через неправильний вибір алгоритмів або формул обчислень. Найпоширенішими є помилки округлення.
7. Помилки управління потоком відбуваються, коли за логікою програми після однієї дії повинна слідувати інша, але виконується зовсім інша дія. Це свідчить про помилки в управлінні потоком команд.
8. Перевантаження виникає через нестачу пам'яті або інших системних ресурсів, що призводить до збоїв у роботі програми.

Контрольні питання до розділу 9

1. Вкажіть принципи і правила створення зручного інтерфейсу.
2. Перерахуйте основні вимоги до компонування форм.
3. Яким чином можна запобігти появі помилок при введенні?
4. Як правильно створити меню?
5. Дайте визначення технологіям тестування «білого ящика» і «чорного ящика».
6. Перерахуйте основні види тестування.
7. Що означає ситуація перегонів? Яким чином її можна протестувати?

РОЗДІЛ 10. ЗАГАЛЬНА МЕТОДОЛОГІЯ ПОШУКУ ОПТИМАЛЬНОГО РІШЕННЯ ІНТЕЛЕКТУАЛЬНИМИ ІНФОРМАЦІЙНИМИ СИСТЕМАМИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ

10.1. Алгоритм визначення структури математичних моделей об'єктів чи процесів

Одним із шляхів підвищення ефективності систем управління та обробки інформації є використання більш точних та достовірних математичних моделей об'єктів чи процесів на основі застосування сучасних методів ідентифікації, що стає можливим із застосуванням досягнень цифрової обчислювальної техніки.

Відомі з літературних джерел підходи до побудови математичних моделей ґрунтуються на використанні регресійного аналізу. При цьому використовується кореляційний, дисперсійний та факторний аналізи, а також аналіз показників суттєвості змінних. Основна мета перерахованих методів полягає у виявленні та виключенні з числа аргументів досліджуваної моделі факторів, що незначно впливають на вихід вихідну змінну моделі.

Крім регресійного аналізу, одним із рекомендованих методів побудови математичних моделей об'єктів є метод групового обліку аргументів (МГУА), що забезпечує порівняно з регресивним аналізом об'єктивний характер моделювання та структурної ідентифікації об'єкта. Об'єктивність досягається тим, що з побудові моделей керуються не заздалегідь заданим числовим значенням окремих обмежень (наприклад, порогового коефіцієнта парної кореляції, критерію суттєвості Стьюдента), а критеріями загального виду: критерію регулярності, мінімуму зміщення та інших.

Як критерій селекції (вибору структури моделі оптимальної складності) нами рекомендовано використовувати критерій мінімуму зміщення $(\eta_{см})$, що дозволяє вирішувати завдання відновлення закону, прихованого в зашумлених експериментальних даних та застосовуваного внаслідок цього для завдання ідентифікації

$$\eta_{см}^2 = \frac{\sum_{i=1}^{N}\left(q_{A_i} - q_{B_i}\right)^2}{\sum_{i=1}^{N} q_{T_i}} \to min,$$

де N - усі точки таблиці вихідних даних; q – вихідна величина; q_A - значення q, розраховані за моделлю , оцінки параметрів якої отримані за точками з великим значенням дисперсії вихідної величини; q_B - те саме, за точками з меншим значенням дисперсії вихідної величини; q_T - Табличне значення зміною.

У МГУА застосовуються дві основні структури генерації безлічі моделей, що оцінюються за критерієм селекції:

1. Комбінаторні (не порогові) алгоритми МГУА.

2. Багаторядні (порогові) алгоритми МГУА.

У першому випадку потрібно завдання, явно ускладненої математичної залежності вихідної величини від вектору вхідних змінних, наприклад, у вигляді полінома високого ступеня, з якої шляхом прирівнювання нулю тих чи інших коефіцієнтів виходять математичні моделі різної структури. Найкраща структура визначається за тим чи іншим критерієм селекції.

У багаторядному алгоритмі спочатку першому ряду селекції утворюються математичні залежності (приватні описи), кожна з яких пов'язує вихідну величину з двома змінними. Отримані приватні описи порівнюються за критерієм селекції, і їх вибирається F_1найкращих. На другому ряду селекції утворюються приватні описи, кожен з яких пов'язує вихідну величину з двома змінними, як виступають приватні описи, отримані на попередньому ряду селекції. З нових приватних описів вибирається F2 найкращих для використання у наступному, третьому ряду селекції. Для кожного ряду є найкраща (за критерієм селекції) модель. Ряди селекції збільшуються, поки оцінка критерію зменшується (правило зупинки).

Розв'язуване нами завдання обумовлено необхідністю розробки досконалішого алгоритму, визначення структури математичної моделі чи процесу.

Поставлена мета визначила необхідність вирішення наступних завдань:

- Розробка комбінованого алгоритму;
- Практична реалізація комбінованого алгоритму у вигляді блок-схеми.

Для вирішення сформульованих завдань використовувалися методи математичного моделювання та програмування.

Пропонований нами модифікований алгоритм МГУА для побудови математичних моделей забезпечує визначення в кожному циклі селекції структуру чіткого опису з використанням для його побудови комбінаторного алгоритму.

У цьому використовуються таблиці поступового ускладнення полінома двох змінних.

Алгоритм представлено на рис. 10.1, де прийняті такі позначення:

1. Початок.

2. Введення експериментальних даних, розрахунок значень функції відгуку за експериментальними даними $\boldsymbol{Y}_i$ та значень аргументів моделі X_{ij} (i = 1,..., N; j = 1,..., M); N – число вимірювань; M – число вихідних змінних моделі).

3. Ранжування точок $\{Y\}$ з дисперсії, виділення навчальної $\{YB_i\}$ та перевірочною $\{YA_i\}$ послідовностей $\{YB_i\}$- Точки з меншим значенням дисперсії вихідної величини; $\{YA_i\}$- Крапка з великим її значенням.

4. Визначення числа окремих описів K:

$$K = C_m^2 = \frac{M!}{2!(M-2)}.$$

5. Початок циклу за рядами селекції: S = 1.

6. Початок циклу за окремими описами.

7. Вибір найпростішої структури для 2-х змінних *G*-го опису.

8. Початок циклу структурами: *L* = 1.

9. Розрахунок параметрів моделі МНК *G*-го опису $\{YA_i\}(A_a)$ і $\{YB_i\}(A_b)$ розрахунок $\eta_{см}$ для *G*-го опису.

10. (*L* = 1)?

11. Визначення критерію для *G*-го опису: $KR_G = \eta_{см}$; L = L+1.

12. Ускладнення структури.

13. $(KR_G \le \eta_{см})$?.

14. Як *G*-го опису приймається структура, що відповідає (*L*-1)-му циклу, розрахунок змінної за *G*-м описом Z_{qi} (*i* = 1,...,*N*); у наступному ряді селекції $\{Z_{qi}\}$ є аргументами приватних описів оптимальної складності

15. (G = K)?

16. G = G + 1.

17. Вибір M найкращих приватних описів по $min\, KR_G$; $KR1_S = min\, KR_G$ (*1, ..., K).*

18. *(S = 1)?.*

19. *S*= S+1.

20. $(KR1_S < KR1_{S-1})$?.

21. Визначення оптимальної структури моделі: як структура оптимальної складності приймається структура моделі з найменшим значенням критерію на попередньому (*S* - 1) – *m* ряду селекції.

22. Кінець.

Використання запропонованого алгоритму забезпечує отримання більш точних та достовірних математичних моделей об'єктів та процесів. Витрати часу для отримання структури математичної моделі знаходяться у функціональній залежності від складності об'єкта або процесу.

10.2. Використання евристичного методу для побудови математичних моделей складних об'єктів керування ІС

Сучасні інформаційні системи, як складні об'єкти управління, характеризуються такими відмінними рисами: наявністю великої кількості вхідних впливів; складним характером залежності між вхідними та вихідними змінними; високим рівнем перешкод; - обмеженим числом датчиків та вимірювальних пристроїв та ін.

У цьому застосування традиційних (балансових чи експериментально - статистичних) моделей керувати об'єктом немає належного ефекту. У цих випадках доцільно використовувати евристичний метод, який використовує для визначення необхідних залежностей та побудови алгоритмів управління, узагальнену думку та формалізації дій кваліфікованих фахівців (експертів).

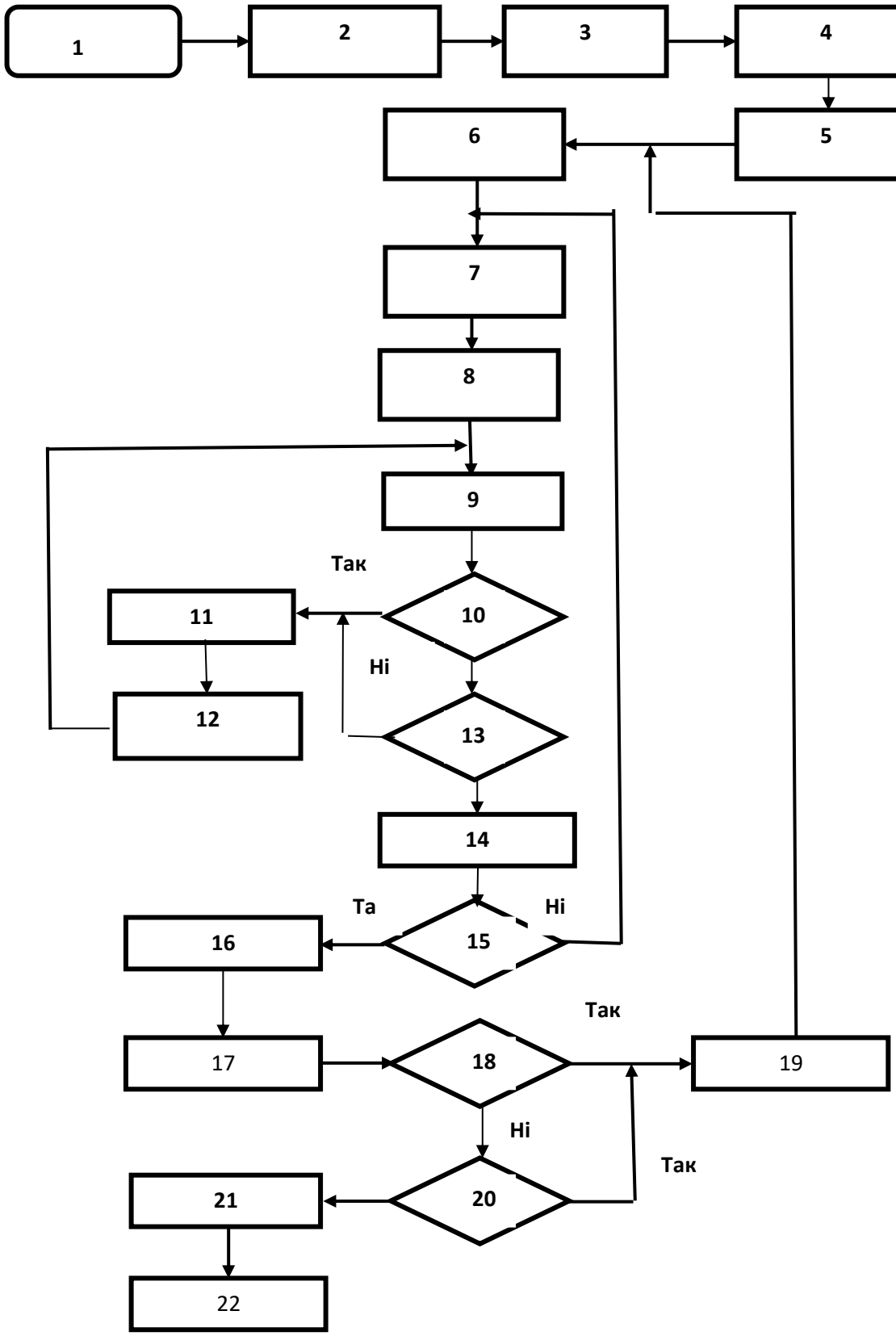

Рис. 10.1. Блок-схема алгоритму визначення структури математичної моделі процесу чи об'єкта

Традиційний метод побудови математичних моделей полягає у вичерпному завданні елементів, що входять до об'єкта управління, та всіх зв'язків між елементами або заснований на проведенні багатофакторного експерименту. Тому основною метою є розробка методу побудови моделі управління у відсутності апріорної інформації про характер залежності між змінними, що входять до її складу.

Для вирішення поставленої задачі використовували методи системного аналізу, статистики та інтегрального обчислення.

Розв'язання задачі побудови математичної моделі складного об'єкта управління засноване на оцінюванні залежності в об'єкті залежності керуючих впливів. $\overline{U^0}$ на інтервалі управління $[\tau]$ від $\overline{X^0}$ $\overline{Y_з^0}$, застосованому на попередніх інтервалах $[\tau-1],[\tau-2]$, ... управління:

$$\overline{Y^0}[\tau]=f\left(\overline{X}^0,\overline{Y}_з^0,\overline{U}^0[\tau-1],\overline{U}^0[\tau-2],...\right)=f\left(V^0\right), \qquad (1)$$

де $\overline{V}^0$ - Вектор стану на $[\tau]$- м інтервалі управління. Індекс "0" відповідає вибірці позитивного досвіду.

Про аналітичний вид функції $f\left(\overline{V}\right)$ відомо тільки те, що вона може бути як завгодно складною.

Введемо деяку оцінку функції (1)

$$\hat{\overline{U}}^0[\tau]=\hat{f}\left(\overline{V}^0\right),$$

і оптимізуватимемо значення параметрів цієї функції за мінімумом середньоквадратичної помилки, тобто будуватимемо таку оцінку, при якій критерій

$$I\left(\overline{U}^0[\tau],\hat{f}\left(\overline{V}^0\right)\right)=\frac{\left[\overline{U}_i[\tau]-\hat{f}\left(\overline{V}_i\right)\right]^2}{n}\to min. \qquad (2)$$

Отже, завдання побудови математичних моделей процесів ІС, реалізованих об'єктом управління, зводиться до процедури конструювання оптимальної оцінки (мінімуму) критерію (2). При побудові моделі, коли апріорна інформація про характер залежності між змінними, що входять до її складу, невідома доцільно застосувати підхід, заснований на використанні непараметричних оцінок умовного математичного очікування.

$$\frac{M\left(\overline{U}[\tau]\right)}{\overline{V}}.$$

Використання умовного математичного очікування як оцінка функції (1) забезпечує мінімум критерію (2), тобто.

$$I\left(\overline{U}[\tau],\hat{f}\left(\overline{V}\right)\right)=\min I\left(\overline{U}[\tau],f\left(\overline{V}\right)\right),\text{ при}\,\hat{\overline{U}}[\tau]=\frac{M\left(\overline{U}[\tau]\right)}{\overline{V}},$$

або в інтегральній формі

$$\overline{U}[\tau]=\int_0^{\tau}\left(\overline{U}[\tau]P\left(\frac{\overline{U}[\tau]}{\overline{V}}\right)\right)d\overline{U}[\tau].$$

Враховуючи що $P\left(\frac{Y}{\overline{X}}\right)=P\frac{\left(Y,\overline{X}\right)}{P\left(\overline{X}\right)}$, умовне математичне очікування перетворимо на вигляд

$$M\left(\frac{\overline{U}[\tau]}{\overline{V}}\right)=\int_0^{\tau}\frac{\overline{U}[\tau]P\left(\overline{V},\overline{U}[\tau]d\overline{U}[\tau]\right)}{P\left(\overline{V}\right)}. \qquad (3)$$

Замість апріорі невідомих функцій щільності ймовірності $P\left(\overline{V}\right)$і $P\left(\overline{V},\overline{U}[\tau]\right)$ використовуємо їх непараметричні оцінки відповідно:

$$\hat{P}\left(\overline{V}\right)=\frac{\left[\sum_{i=1}^{n}P^*\left(\overline{V},\overline{V}_i\right)\right]}{n}, \qquad (4)$$

$$\text{і}\,P\left(\overline{V},\overline{U}[\tau]\right)=\frac{\left[\sum_{i=1}^{n}P^*\left\{\overline{V},\overline{U}[\tau],\overline{V}_i,\overline{U}_i[\tau]\right\}\right]}{n}. \qquad (5)$$

Відомо, що такі оцінки асимптотично сходяться до $P\left(\overline{V}\right)$і $P\left(\overline{V},\overline{U}[\tau]\right)$за умови, що функції вкладів n $P^*\left(\overline{V},\overline{V}_i\right)$, i=1,2,…, симетричні щодо відповідних векторів $\overline{V}_i$ (кожен з них є математичним очікуванням відповідної йому функції вкладів), невід'ємні, спадають при $n\to\infty$ та одиничної площі. Крім того, розмах цієї функції асимптотично зменшується в міру збільшення n. Асимптотична збіжність оцінок (1) і (5) зумовлює асимптотичну збіжність тих, що використовують ці оцінки, непараметричну конструкцію умовного математичного очікування у формі (3).

Введемо функцію вкладів виду

$$P^*\left(\overline{V},\overline{V}_i\right)=\left[\frac{1}{\sigma_i^m}\left(2\pi\right)^{\frac{m}{2}}\right]exp\left[\frac{-d^2\left(\overline{V},\overline{V}_i\right)}{2\sigma_i^2}\right],\ i=1,\,2\ ...,\,n \qquad (6)$$

$$\text{і}\,P^*\left\{\left(\overline{V},\overline{U}[\tau]\right),\left(\overline{V}_i,\overline{U}_i[\tau]\right)\right\}=\left[\frac{1}{\sigma_i^{m+1}\left(2\pi\right)^{\frac{(m+1)}{2}}}\right]exp\left(-d^2\frac{\left\{\left(\overline{V},\overline{U}[\tau]\right),\left(\overline{V}_i,\overline{U}_i[\tau]\right)\right\}}{2\sigma_i^2}\right).$$

Враховуючи що $d^2\left\{\overline{V},\overline{U}[\tau],\overline{V}_i,\overline{U}_i[\tau]\right\}=d^2\left(\overline{U}[\tau],\overline{U}_i[\tau]+d^2\left(\overline{V},\overline{V}_i\right)\right),\ i=1,\,2\ ...,\,n$,

Виразимо $P^*\{(\overline{V},\overline{U}[\tau]),(\overline{V}_i,\overline{U}_i[\tau])\} = P^*(\overline{V},\overline{V}_i)P^*(\overline{U}[\tau],\overline{U}_i[\tau])$.

Введемо у формулу (3) замість невідомих функцій $P(\overline{V})$ і $P(\overline{V},\overline{U}[\tau])$ їх непараметричні оцінки (4) та (5). Враховуючи, що інтегрування ведеться за змінною $\overline{U}[\tau]$, отримуємо

$$\overline{U}[\tau] = \frac{\left\{\sum_{i=1}^{n} P^*(\overline{V},\overline{V}_i)\int_0^{\tau}\overline{U}[\tau]P^*(\overline{U}[\tau],\overline{U}_i[\tau])d\overline{U}[\tau]\right\}}{\Sigma P^*(\overline{V},\overline{V}_i)}. \qquad (7)$$

У формулі (7) і - й інтеграл є математичним очікуванням змінної $\overline{U}[\tau]$, розподіленою відповідно до $P^*(\overline{U}[\tau],\overline{U}_i[\tau])$, для якої змінна $\overline{U}_i[\tau]$ є математичним очікуванням.

Отже,

$$\int_0^{\tau}\overline{U}[\tau]P^*(\overline{U}[\tau],\overline{U}_i[\tau])d\overline{U}[\tau] = \overline{U}_i[\tau],\ i = 1,2,\ ...,\ n, \qquad (8)$$

Підставляючи (8) та (7), отримуємо наступне вираз для моделі процесу управління на $[\tau]$- м інтервалі управління:

$$\hat{\overline{U}}^0[\tau] = \frac{\sum_{i=1}^{n} U_i[\tau]P^*(\overline{V}^0,\overline{V}_i)}{\Sigma P^*(\overline{V}^0,\overline{V}_i)}, \qquad (9)$$

де $\overline{U}_i(\tau)$- значення керуючих впливів, застосованих при виконанні реалізованого процесу вибірки позитивного досвіду $[\tau]$-м інтервалі управління, а функція вкладу $P(\overline{V}^0,V_i)$ визначено формулою (6).

Незважаючи на загальну непараметричну природу математичних моделей $\hat{\overline{U}}^0[\tau]$ і $\overline{U}[\tau]$, алгоритми, що використовують їх виходять різними- перша модель враховує всі управління, застосовані на $[\tau]$-м інтервалі вибірки позитивного досвіду з властивим кожному процесу вибірки значенням вагового коефіцієнта; друга модель зводиться до пошуку у вибірці позитивного досвіду єдиної та найбільш схожої до цього $[\tau]$-му моменту раніше проведеного процесу.

10.3. Модифікований алгоритм глобального статистичного пошуку екстремуму оптимізованих функцій з дискретною зміною їх аргументів

У зв'язку з переходом на економічні методи управління народним господарством намічено новий перспективний етап комплексної автоматизації промислових виробництв на основі використання інтегрованих інформаційних систем. Поряд із завданнями управління доводиться вирішувати також і оптимізаційні завдання. Подальше розширення робіт з

розроблення та впровадження інформаційних систем пов'язано, перш за все, зі збільшенням оптимізаційних завдань.

При оптимізації процесів інформаційних систем, параметри яких можуть набувати лише чисельних чи дискретних значень, виникають труднощі, які полягають у тому, що алгоритми, що моделюють функціонування виробничих систем, дозволяючи отримувати чисельні значення їх характеристик та обраних критеріїв оптимальності лише в окремих точках, не дають можливості оцінити характер функції загалом. Викладені обставини практично виключають можливість застосування класичних підходів, що ґрунтуються на диференціюванні методів оптимізації для оптимізації виробничих процесів, що описуються за допомогою таких моделей.

Найбільш доцільно вирішувати задачі оптимізації описаних моделей чисельними методами, такими як статистичний пошук. Для цих цілей необхідно розробити алгоритм глобального статистичного пошуку екстремуму оптимізованих функцій з дискретною зміною аргументів, який би забезпечував оптимальне рух системи у напрямку пошуку нових чергових локальних екстремумів з урахуванням передісторії руху.

Загальна ідея спрямованого статистичного пошуку полягає в тому, що при знаходженні системи у стані X_i у просторі оптимізованих параметрів робиться крок у випадковому напрямку ε. Якщо значення оптимізується, у новому стані (у новій точці пошуку) X_{i+1}, ближче до екстремуму, ніж значення цієї функції в попередній точці X_i, то система переводиться в новий стан X_{i+1} і наступний крок у випадковому напрямку робиться з попередньої точки X_i.

Основною інформацією, необхідної для статистичної оптимізації, є значення критерію ефективності окремих точках пошуку, т. е. саме ті величини, які можна отримати при моделюванні методом Монте-Карло.

Для виконання кроку у випадковому напрямку суттєво важливим є поняття k*-вимірний випадковий вектор. Під одиничним випадковим вектором ε у просторі параметрів мається на увазі, наступний одиничний вектор, окремі реалізації якого спрямовані: рівноймовірно у всіх напрямках простору параметрів, тобто цей вектор рівномірно розподілений по всіх напрямках простору параметрів.

Існує кілька методів реалізації випадкового вектору. Доцільним буде обрати такий метод реалізації випадкового вектора ε, коли його проекціями на осі координат будуть незалежні нормально розподілені випадкові числа. Якщо отримати k* незалежних випадкових значень нормально розподіленої величини $x1, x2, ..., x_k$* , то напрямок випадкового вектора X, побудованого цих величинах як у складових буде рівномірно розподіленим у сфері (тобто, на безлічі всіх можливих напрямів векторів в k* -мерном просторі.

Оскільки випадковий вектор визначений як одиничний вектор, це твердження є рівнозначним тому, що проекції випадкового вектора X рівномірно розподілені на поверхні гіперсфери одиничного радіусу*(R=1).*

Для локального статистичного пошуку оптимізації систем управління використовуються метод пошуку з покаранням випадковістю, метод статистичного градієнта, метод найкращої проби.

Основна ідея випадкового пошуку методом статистичного градієнта полягає в тому, що напрямок чергового кроку пошуку вибирається за статистичним градієнтом або, що те саме, у напрямку середньозваженого збільшення функції якості, отриманого в результаті ряду випадкових проб.

Сутність оптимізації з використанням пошуку з покаранням випадковістю полягає у наступному. У просторі оптимізованих параметрів робиться i-й крок у випадковому напрямку ε. Якщо значення функції у новому стані $W(X_{i+1})$ більше (при пошуку максимуму функції), ніж значення $W(X_i)$ у вихідному стані, повторюється крок у тому ж напрямку ε. Якщо ж значення функції у новому стані $W(X_{i+1})$ не збільшилося, то система повертається в початковий стан, звідки робиться наступний пошуковий крок у новому випадковому напрямку, тобто невдача «карається» новою випадковістю.

Статистична оптимізація функції $W=W(X)$ за алгоритмом пошуку методом найкращої проби полягає в наступному. У просторі оптимізованих параметрів із вихідної точки X_i робиться j^* випадкових проб (пробних кроків) $g\varepsilon_1, g\varepsilon_2, ..., g\varepsilon_{j^*}$ і вибирається напрямок ε^{++}, що призводить до максимального зниження (при пошуку мінімуму) цільової функції

$$W\left(X_i + g\varepsilon^{++}\right) = \min_{(j)}\left\{W\left(X_i + g\varepsilon_j\right)\right\},$$

де $j = 1,2,..., j^*$ - Номер проби; g - масштаб пробного кроку пошуку.

В цьому напрямку ε^{++} робиться ряд робочих кроків до продовження приватного оптимуму в цьому напрямі, після чого виконується процедура вибору нового напряму випадкового пошуку до перебування оптимуму.

Істотним недоліком розглянутих вище статистичних методів локальної оптимізації полягає у значних витратах машинного часу через необхідність повторного повернення системи у вихідну точку X_i, у разі віддалення від екстремуму, та подальшого вибору нових кроків до досягнення нового стану X_{i+1}, що наближає функцію, що оптимізується, до екстремуму.

Завдання управління виробництвом полягає у синтезі системи управління, що забезпечує найкраще у заданому сенсі наближення фактичної траєкторії руху об'єкта управління y_t до бажаної. Ціль управління y'_t, обмеження на режим руху до мети Y, U та вид показника якості I, Яким оцінюється якість управління, задані ззовні системою вищого рівня управління.

Відхилення фактичної траєкторії руху від бажаної може виникати як

внаслідок зовнішніх збурень, що діють, так і через зміни заданої траєкторії. y_t^*

Сутність запропонованого методу полягає в тому, що вибір чергової точки пошуку оптимуму здійснюється не рівноймовірно, а з деякими обмеженнями, внаслідок чого рух системи набуває інерційність і вона проходить повз локальні екстремуми, відзначаючи лише області їх існування. Пройшовши повз чергову область локального екстремуму, система шукає такі найкращі напрями пошуку відносного нових екстремумів, а чи не повертається до вже дослідженим. При цьому траєкторія руху системи в процесі пошуку вибирає оптимально в напрямку чергового локального екстремуму з урахуванням передісторії руху.

Нехай показник ефективності функціонування виробництва як складної системи має вигляд

$$W = W\left(x_1, x_2, ..., x_{k^*}\right),$$

де W - Вибраний показник ефективності системи; k^* – загальна кількість параметрів системи.

Нехай ця функція задана в області $\{D\}$, що є k^*-мірним простором, обмеженим k^*-мірним інтервалом $x_{k_{\min}} \le x_k \le x_{k_{\max}}$, $k = 1,2,...,k^*$ на функцію $W(X)$ накладено обмеження виду: $R_\lambda\left(X\right) \le b_\lambda$, $\lambda = 1,2,...,\lambda^*$, де b_λ - Відповідне граничне значення.

Для знаходження оптимальної системи потрібно знайти вектор $X^* = \left(x_1^*, x_2^*, ... x_k^*\right)$, при якому показник ефективності системи набуває мінімального (максимального) значення на безлічі допустимих векторів $W^* = \text{extr}\ \{W(X)\}$.

Обмеження області вибору чергової точки пошуку здійснюється в такий спосіб (рис. 19).

Нехай $П = \left(П_1, П_2, ... П_{k^*}\right)$- Вектор пам'яті, напрям якого відповідає максимальній зміні функції якості в попередній період пошуку локального екстремуму. Вибиратимемо точки пошуку, рівномірно розподілені на поверхні k^* - мірної гіперсфери в межах області обмеженої перетином гіперсфери в межах області, обмеженої перетином гіперсфери та k^* - мірного гіперконусу з віссю $П$ та кутом розкриття 2ψ .

Гіперсфера висунута від початку координат у напрямку $П$ на відстань $|П|$. Точки пошуку визначаються вибором випадкового спрямування $\varepsilon = \left(\varepsilon_1, \varepsilon_2, ..., \varepsilon_{k^*}\right)$.

$$\varepsilon_k = \frac{П_k + gc\varepsilon_k^o}{\left|\sqrt{\sum_{k=1}^{x^*}\left(П_k + c\varepsilon_k^o\right)^2}\right|}, \quad k = 1,2,...k^* \tag{10}$$

$\psi = \arcsin\left\lfloor gc - \varepsilon_o \right| / |\Pi| \rfloor$, де $\varepsilon_o^* = \left(\varepsilon_{01}, \varepsilon_{02}, ... \varepsilon_{0k^*} \right)$- Поодинокий випадковий вектор, рівномірно розподілений по всіх напрямках простору параметрів; $c \le 1$- Постійна, що визначає радіус гіперсфери.

Для $k^* = 2$, $g = 1$, $c = 0{,}8$ ситуація показано на рис. 10.2.

Напрямок чергового робочого кроку, що найвигідніше розташований у обраному гіперконусі пошуку, можна вибрати за допомогою методу статистичного градієнта або методу найкращої проби. У цьому випадку цей вибір проводиться методом статистичного градієнта, основна ідея якого полягає в тому, що напрямок чергового кроку пошуку вибирається за статистичним градієнтом або, що те ж саме, за напрямом середньозваженого збільшення функції якості, отриманого в результаті ряду випадкових проб.

Якщо $w(X)$– функція, яка залежить від векторного аргументу та у просторі оптимізованих параметрів з вихідної точки X_i робиться j^* випадкових проб, то k-я відповідна вектор Y (вектору, що визначає напрямок і довжину чергового робочого кроку пошуку) найвигіднішим чином розташованого у просторі пошуку матиме вигляд

$$y_k = \sum_{j=1}^{j^*} \varepsilon_{k\,j} \Delta W_j, \qquad (11)$$

де $j = 1,2,..., j^*$ - Номер чергової проби для вибору найкращого напряму пошуку; $\varepsilon_j = \left(\varepsilon_{1j}, \varepsilon_{2j}, ... \varepsilon_{k^* j} \right)$– випадковий вектор, сформований для j-ї проби; ε_{kj}- псевдовипадкове нормально розподілене число з нульовим математичним очікуванням та одиничною дисперсією; $\Delta W_j = W\left(X + g\varepsilon_j\right) = W\left(X_j\right)$- Збільшення функції при j-ї пробі; g- масштаб пробного кроку пошуку.

Довжина вектору Y визначається виразом $|Y| = \left| \sqrt{\sum_{k=1}^{k^*} (y_k)^2} \right|$, та напрямок k-ї складової вектору Y визначається як

$$\cos\varphi_k = y_k / |Y| \qquad (12)$$

Вибір вказаного напрямку y_k при $k^* = 2$ показаний графічно на рис. 10.2. При цьому $y_1 = \varepsilon_1^{(1)} \Delta W_1 + \varepsilon_1^{(2)} \Delta W_2 = \sum_{j=1}^{2} \varepsilon_k^{(j)} \Delta W_j$

Пропонований алгоритм глобального пошуку подано на рис. 10.4а 10.4б.

Оператор 1 здійснює введення вихідної інформації: $w(X)$- Функція, що підлягає оптимізації (вводиться підпрограмою); $X_0 = \left(x_{10}, x_{20}, ... x_{k^* 0} \right)$- Вихідна точка пошуку; k^*- Число параметрів, піт яким проводиться оптимізація;

$\left\{x_{k_{\min}}, x_{k_{\max}}\right\}, k = 1,2,...,k^*$ – межі області завдання оптимізованої функції; c - Постійна визначальна радіус гіперсфери глобального пошуку; $\{\Delta k\}, k = 1,2,...,k^*$ - Інтервали дискретності за параметрами; g - масштаб пробних кроків; j^* - Число пробних кроків.

Оператор 2 констатує факт відсутності передісторії.

Оператори 3 та 4 здійснюють вибір вихідної точки пошуку $X_i(i=0)$ розрахунок значення функції якості $W(X_i)$ для цієї точки.

Оператор 5 починає вибір найкращого напрямку пошуку в межах обраного гіперконусу.

Оператор 6 здійснює вибір для i-го кроку та j-ї проби випадкового вектора $\varepsilon_i^{(j)}$, рівномірно розподіленого у просторі параметрів.

Оператори 7-9 унормують обраний вектор.

Оператори 10-12 формують випадковий вектор у межах обраного гіперконусу виходячи з умови (36).

Оператори 13–18 здійснюють у циклі перевірку відомих обмежень типу xki min≤*x(j) k(i+1)*≤*xkmax*та повернення точки в задану область {*D*}

Оператор 19 розраховує значення функції якості в новій $(i+1)$-й точці для j-ї проби.

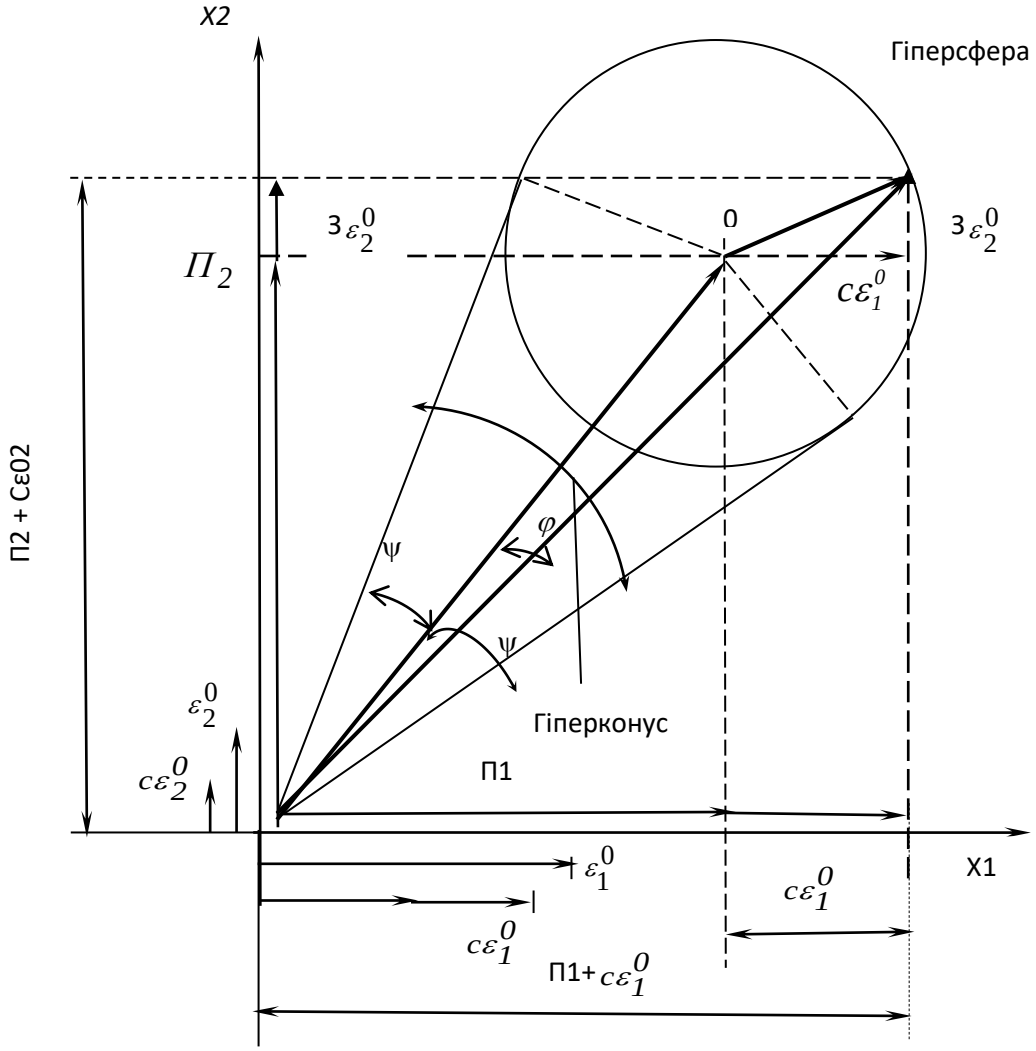

Рис. 10.2. Напрямний конус та напрямна сфера

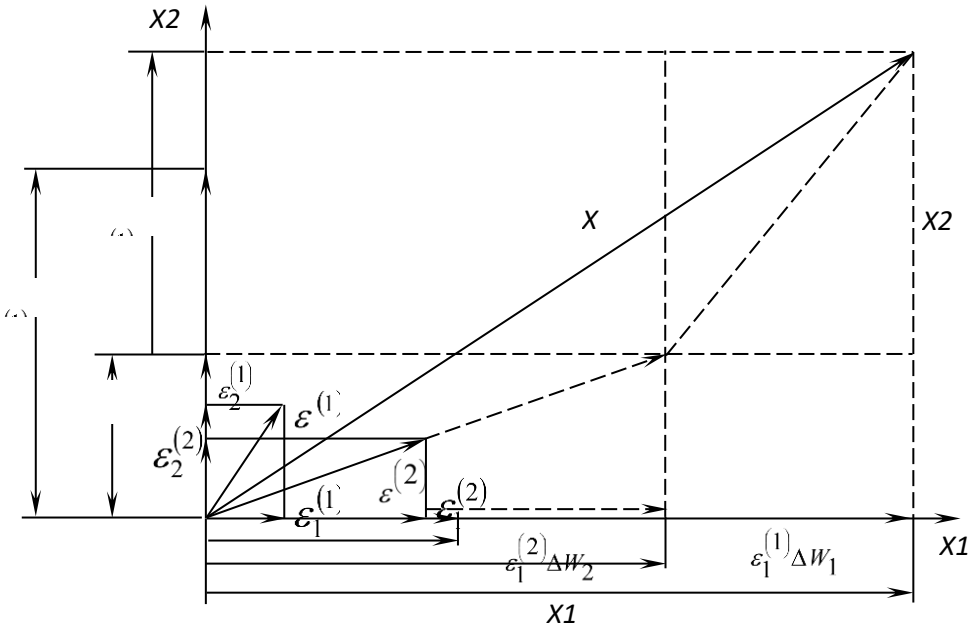

Рис. 10.3. Вибір напряму за статистичним градієнтом

Оператори 20-21 здійснюють накопичення векторної суми. y_k за формулою (13).

Група операторів 6–22 працює j^* раз за кількістю випадкових проб, після чого оператори 24-25 розраховують $\cos\alpha_k$ за формулою (11)

Оператор 28 здійснює вибір величини робочого кроку з самоналаштуванням: у міру наближення до локального екстремуму кут розкриття конуса зменшується, щоб система не «зав'язла» в цьому локальному екстремумі, а пройшла повз, відзначивши його область, і навпаки, у міру віддалення системи від локального локального екстремуму кут розкриття конуса збільшується, що підвищує ефективність пошуку у новій ще не дослідженій частині області

$$\psi_{\text{пі}} = \begin{cases} 1, если\ приращение функции\ положительно\ (знак "+"), \\ 0, в\ противном\ \ случае (знак "-"). \end{cases}$$

Якщо знак збільшення міняється з плюсу на мінус, то очевидно, що система проходить локальний максимум, якщо з мінуса на плюс - локальний мінімум. У цих випадках оператори 42 і 43 фіксують характер екстремуму функції якості (*min* і *max*) координати точок пошуку та номера N, відповідних пройденим областям локальних екстремумів, після чого оператор 44 здійснює перевірку на закінчення пошуку за спеціальною програмою.

Цьому оператору передається управління і після операторів 37 та 41 у випадках, коли знак ΔW не змінюється, тобто коли система не проходить екстремальної області. Пошук закінчується, коли система проходить всю область завдання функції вздовж одного з параметрів або виконує задану кількість кроків пошуку.

Пропонований метод заснований на використанні відомого алгоритму глобального статистичного пошуку по напрямній сфері. Алгоритм модифікований для оптимізації функцій з дискретною зміною аргументів і внесено додаткові вдосконалення, які прискорюють процес пошуку.

При його використанні вибір чергової точки пошуку оптимуму здійснюється не рівноймовірно, а з деякими обмеженнями, внаслідок чого рух системи набуває інерційності, і вона проходить повз локальні екстремуми, відзначаючи лише області їх існування.

10.4. Алгоритми оптимізації багатокритеріальних завдань прийняття рішень із нечіткими цілями

Перехід до ринкових методів господарювання призвів до переведення великої кількості інформаційних систем із державного бюджету до розряду самостійних суб'єктів, перед якими виникає набагато більше проблем прийняття раціональних рішень.

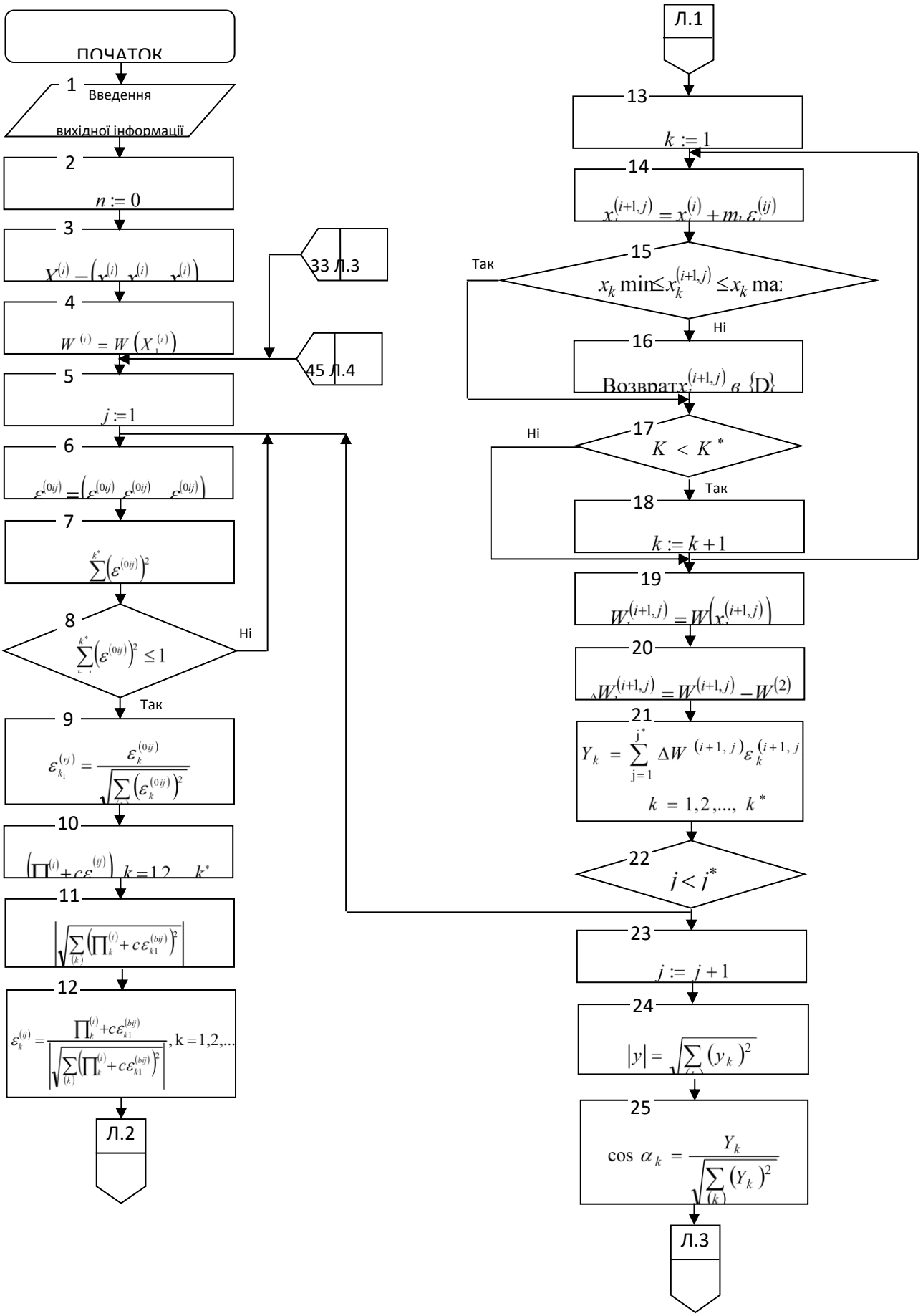

Рис. 10.4а. Алгоритм глобального статистичного пошуку

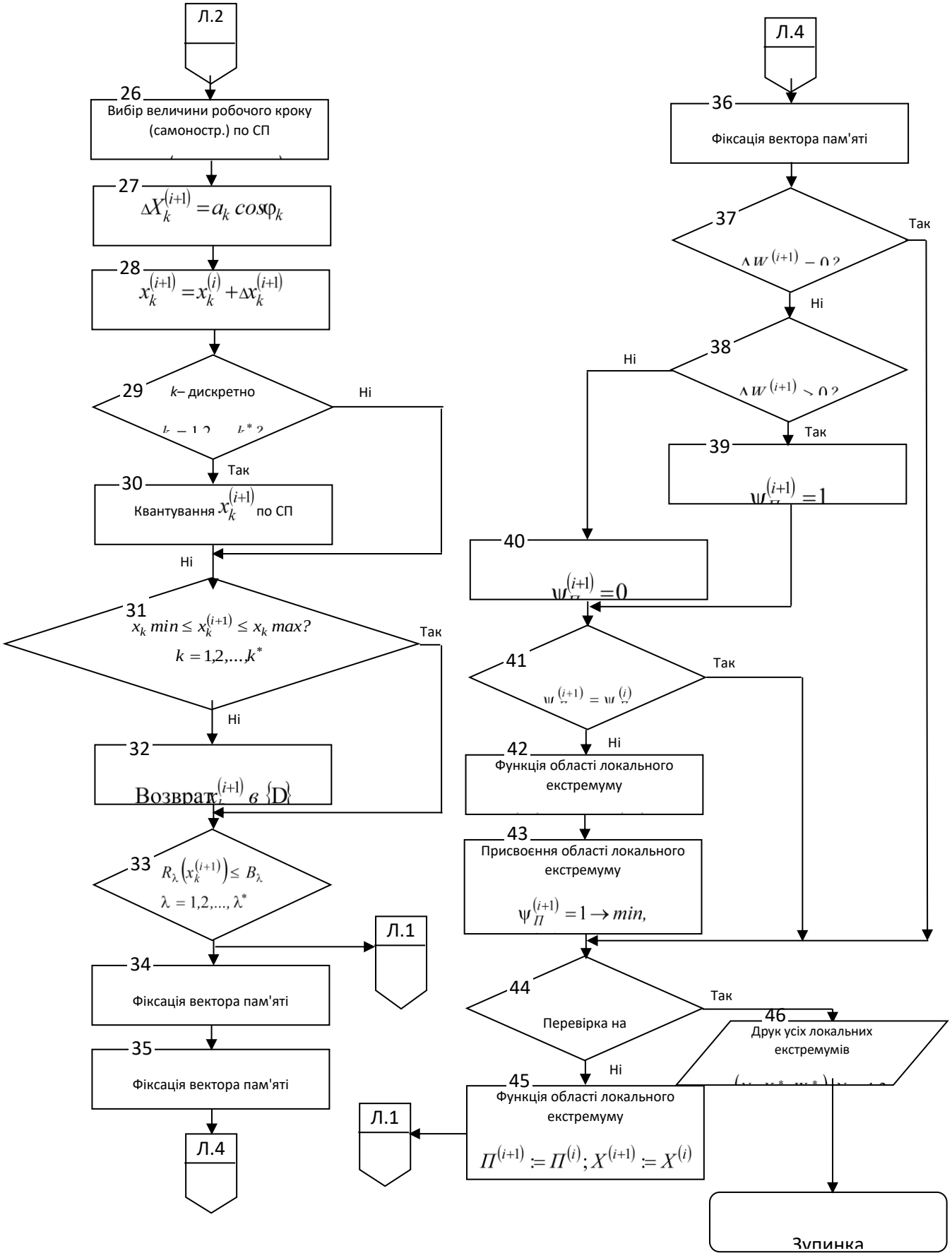

Рис. 10.4б Алгоритм глобального статистичного пошуку (продовження)

Величезні масиви інформації, що переробляється, відсутність спеціалізованих стійких алгоритмів, велика відповідальність при прийнятті рішень призводять до необхідності вирішення нових завдань, пов'язаних з розробкою математичних моделей, наукових методів, які дозволяють підвищити ефективність функціонування інформаційних систем, що функціонують в умовах невизначеності, що виникає внаслідок неадекватності моделей, мети та критеріїв функціонування окремих частин інформаційної системи, прояви зовнішнього середовища. Тому проблема підвищення ефективності та якості підтримки прийняття рішень в управлінні інформаційною системою на даний час залишається актуальною і набуває ще більшої гостроти.

Попередній аналіз цієї проблеми показує, що, по-перше, методи прийняття рішень багато в чому визначаються специфікою завдань та способом їх формалізації; по-друге, відсутні дослідження, пов'язані з використанням нечіткого вибору альтернатив за умов невизначеності. Проведений аналіз відомих моделей показав їхню малопридатність для вирішення завдань управління в інформаційній системі.

Формально багатокритеріальна задача прийняття рішень з нечіткими цілями може бути описана такими елементами: - множиною допустимих альтернатив $X \subset R^n$ з елементами x; - безліччю всіх можливих результатів, оцінок альтернатив $X \subset R^m$; - функцією критерію $f(\cdot) :\to Y$, що встановлює зв'язок між альтернативами та їх оцінками: $f(x) = (f_1(x),..., f_m(x))$, $f_i(\cdot): X \to R^1, i = 1,2,..., m$; - функціями приналежності $\mu_{c_i} : R^1 \to I, i = 1,2,..., m$, де I - одиничний відрізок числової прямої, що характеризує переваги особи, яка приймає рішення.

Відповідно до [10] під оптимальним рішенням завдання прийняття рішення з нечіткими цілями розуміється альтернатива

$$x^* \in X : x^* = arg\, max\, \mu_D(x), \tag{13}$$

де функція приладдя $\mu_D(x)$ визначається формулою:

$$\mu_D(x) = \bigwedge_{i=1}^{m} \mu_{c_i}(f_i(x)), \tag{14}$$

або

$$\mu_D(x) = \sum_{i=1}^{m} \lambda_i \mu_{c_i}(f_i(x)), \lambda_i > 0, \ \sum \lambda_i = 1. \tag{15}$$

У формулі (15) числа λ_i грають роль коефіцієнтів відносної важливості цілей. Надалі під функціями розумітимемо звуження цих функцій на безлічі

$\overline{sup C_i}$, тобто

$$\mu_{c_i} = \mu_{c_i}\left|\overline{sup C_i}\right., \tag{16}$$

де $\overline{sup C_i}$ є замикання безлічі $sup C_i = \left\{ f_i(x) \in R^1 \middle| \mu_{c_1}(f(x)) > 0 \right\}$.

Припустимо, що безліч $sup C_i$ є пов'язана безліч у R^1 тобто замкнутий інтервал $[r_i, R_i]$. При цьому припущенні умова (17) еквівалентна наступній системі нерівностей:

$$r_i \le f_i(x) \le R_i,\ i = 1,2,...,m. \tag{17}$$

Таким чином, пошук альтернативи, яка задовольняє умови еквівалентний розв'язанню задачі математичного програмування:

$$\mu_D(x) \to max, \tag{18}$$

$$r_i \le f_i(x) \le R_i,\ x \in \mathrm{X}. \tag{19}$$

Оскільки основним представником завдань прийняття рішень, за умов невизначеності початкових даних, є завдання лінійного програмування, ми обмежимося розглядом випадку, коли функції $f_i(x)$ лінійні: $f_i(x) = \sum_{j=1}^{n} x_i \alpha_{ij}$.

Нехай коефіцієнти лінійних функцій є нечіткими та характеризуються функціями приналежності $\mu_{Ai}(\alpha_i): R^n \to I$.

Користуючись поняттям умовної нечіткої множини, визначимо нечітку множину $U_i \subset R^n$ з функцією приладдя

$$\mu_{Ui}(x) = \sup_{\alpha}\left(\mu_{Ai}(\alpha_i) \wedge \mu_{C_i}(f_i(x, \alpha_i))\right). \tag{20}$$

З урахуванням нечіткості початкових даних функція власності $\mu_D(x)$ визначається формулою $\mu_D(x) = \bigwedge_{i=1}^{m} \mu_{Ui}(x)$, або $\mu_D(x) = \sum_{i=1}^{m} \lambda_i \mu_{Ui}(x)$.

Розглянемо задачу (21) для випадку, коли коефіцієнти лінійних функцій з невзаємодіючими нечіткими множинами. У цьому умовні функції власності $\mu_{Ai}(\alpha_i)$ допускають уявлення $\mu_{Ai}(\alpha_i) = \bigwedge_{j=1}^{m} \mu_{Aij}(\alpha_{ij})$, де $\mu_{Ai}(\alpha_{ij})$- функція власності виду $\mu_{Aij}: R^1 \to I$.

Нехай, крім цього, при $i \in [1:m]$, $j = [1:n]$ $\mu_{Aij}(\alpha_{ij}) = \begin{cases} 1, & \alpha_{ij} \in [\underline{\alpha}_{ij}, \overline{\alpha}_{ij}] \\ 0, & \alpha_{ij} \notin [\underline{\alpha}_{ij}, \alpha_{ij}] \end{cases}$

При цих обмеженнях формула (21) набуває вигляду

Введемо безліч $K_j \subset R^m : K_j = \{\alpha_i \in R^m | \underline{\alpha}_{ij} \le \alpha_{ij} \le \overline{\alpha}_{ij},\ i = 1,2,...,m\}$.

Через нечіткість коефіцієнтів лінійних функцій система обмежень може бути описана у вигляді $x_1K_1 + x_2K_2 + ... + x_nK_n \subseteq K(r,R)$, де $K(r,R) = \{y \in R^m | r \le y \le R\}$.

При $x \ge 0$ завдання математичного програмування $\mu_D(x) \to max$, $\begin{cases} x_1K_1 + x_2K_2 + ... + x_nK_n \subseteq K(r,R) \\ x \in X,\ x \ge 0 \end{cases}$, Еквівалентна задачі (14).

Вироблення управлінських рішень доцільно проводити у межах діалогової системи. Вся інформація має зберігатися у БД.

Цінність інформації є основною характеристикою для ухвалення управлінського рішення і може бути виражена через приріст ймовірності оптимального рішення. Якщо для отримання інформації ймовірність оптимального рішення була рівна P_0, а пізніше постійна P_1, то цінність $V = log_2 P_1 - log_2 P_0 = log_2\left(P_1 / P_0\right)$.

p align="justify"> Методи управління технологічними комплексами при створенні локальних систем управління повинні враховувати інформаційні зв'язки між окремими підсистемами цього технологічного комплексу. Для обчислення оцінки економічності прийняття рішень використовуємо такий вираз $U_y = \sqrt[3]{U_t, U_x, U_э}$, де U_y - Оцінка керівника рішення; U_t - Комплексна оцінка ефективності праці (співвідношення досягнутої ефективності роботи до базисної); U_x - господарський ефект (відносини досягнутої величини доходу до базисної); $U_э$ - економічність (відносини віддачі витрат за управління до базисної).

Для ієрархічних систем управління виробництвом загальні закономірності прийняття рішень можна сформулювати у вигляді функціонального принципу теорії прийняття рішень – принципу послідовного вирішення невизначеності. Даний принцип характеризує рух від загального уявлення про мету, характер діяльності, вимогу функціонування та розвитку керуючої системи про показники її раціональної роботи як цілісності до детального подання завдань. У процесі цього руху кожному рівні уявлення системи, починаючи з вищого, з безлічі допустимих альтернативних рішень для подальшого розгляду відбираються ті, які заслуговують уваги з цілей системи, інші відкидаються і більше розглядаються.

Правильність вибору альтернатив кожному рівні узагальнення перевіряється шляхом їх аналізу кожному, більш детальному рівні представлення стану системи, завдяки чому початкові альтернативи

уточнюються, які кількість скорочується. З'являється також можливість кількісної оцінки початкової невизначеності завдання прогнозу $Еполн$ та ступеня її вирішення $Еост$, які можна обчислити: $E_{полн} = E_{нач} + E_T + E_{остат}$, де $Еост$ - Залишкова невизначеність рішення; E_T - невизначеність інформації, яка вирішується логічними методами; $Енач$ - Початкова невизначеність рішення $E_{ост} = f(I(+))$, де $I(+)$ - Глибина прогнозу.

Разом з $Еполн$ і $Еост$ зручно користуватися відносною оцінкою міри якості рішення $R = 1 - \left(\frac{E_{ост}}{E_{полн}} \right)$.

З практичного погляду значення міри якості R полягає в тому, що з її допомогою з'являється можливість порівняння та оцінки ефективності прийнятих рішень або їх окремих елементів.

10.5. Синтез стратегії управління задовольняє заданому критерію адаптації та мінімуму критерію якості перехідного процесу

Перспективним напрямом в автоматизації інформаційної системи є створення систем, що дозволяють активно здійснювати адаптацію до умов, вимог, що змінюються, цілей.

В результаті аналізу опублікованих робіт, присвячених адаптивним системам, нами встановлено, що в них розглядаються питання адаптивної комутації з пріоритетом, адаптивної дискретизації з використанням методу найменших квадратів та експоненційних функцій, адаптивної ідентифікації, синтезу банку алгоритмів адаптації, з метою визначення найбільшого сімейства законів адаптації, що гарантують стійкість адаптивних систем в цілому, серед яких слід вибрати найкращий у сенсі деякого критерію якості.

Серед перелічених та інших відомих робіт відсутні роботи присвячені синтезу стратегії управління що задовольняє критерію адаптації і мінімуму якості перехідного процесу.

Розглянемо деяку систему, на яку впливають у кожний момент часу $t = k$ (час t - дискретно) некерований вплив S_k (загалом вектор) і керований вплив U_k (загалом вектор). Вплив S_k, $k = 1,2, \ldots$ є послідовністю незалежних однаково розподілених значень випадкової величини $S \in dF(S/\Theta)$, де $F(S/\Theta)$ - функція розподілу ймовірностей S, що належить параметричному сімейству $\{F(S/\Theta),\ \Theta \in \Omega\}$.

Функціональна форма $F(S/\Theta)$ - відома, а параметр $\theta \in \Omega$ (загалом – вектор однозв'язкової області Ω у евклідовому просторі) невідомий. Позначимо через $\varphi(S_k, U_k)$ деяку функцію, що визначає стан поточних втрат системи в k - ом такті. Введемо середню характеристику стану поточних втрат:

$$\bar{\varphi}(U_k,\Theta)=\underset{S_k}{M}\{\varphi(S_k,U_k)\}=\int_{S_k}\varphi(S,U_k)\,f(S/\Theta)dS,$$

де $M\{\cdot\}$- оператор математичного очікування щодо випадкової величини S_k; $f(S/\Theta)$- Щільність розподілу ймовірностей випадкової величини S.

Протягом певного часу відбувається накопичення інформації у формі результатів спостережень за некерованим впливом S_k.

Нехай S^k - Вектор з компонентами $(S_1,S_2,\ldots,S_k)$- вибірка спостережень обсягом k. Вектор S^{k-1} містить всю інформацію, на основі якої потрібно зробити вибір U_k. Будемо називати S^{k-1} інформаційним станом системи на момент часу k, а функцію виду $U_k=V_k(S^{k-1})$- Законом управління в k- м такті. Послідовність законів управління $\pi=\{V_1,\ldots,V_k,\ldots,V_N\}$ назвемо стратегією управління, де N - Довжина інтервалу управління.

Систему вважаємо адаптивною по відношенню до моменту l, якщо з цього моменту, її алгоритм управління виробляє таку стратегію $\pi_l=\{V_l,\ldots,V_k,\ldots,V_N\}$, яка перекладає при $N\to\infty$ систему зі стану середніх очікуваних втрат $\bar{\varphi}(V_l(S^{l-1}),\Theta)$ у стан $\bar{\varphi}(U^0(\Theta),\Theta)$- стан мінімальних очікуваних втрат при оптимальному управлінні $U^0(\Theta)$, що визначається в умовах наявності повної інформації про параметр $\Theta\in\Omega$. Збіжність $\bar{\varphi}(V_l(S^{l-1}),\Theta)$ до $\bar{\varphi}(U^0(\Theta),\Theta)$ у згаданому вище імовірнісному сенсі є критерієм адаптації. З визначення адаптивності системи слід, що у цьому шляху можна побудувати безліч стратегій управління, у яких система буде адаптивної, т. е. задовольняти заданому критерію адаптації. Проте кожній стратегії π_l відповідають свої очікувані втрати перехідного процесу $W_N(\pi_l)=\sum_{k=l}^{N}\underset{S^{k-1}}{M}\{\Phi(\bar{\varphi}(V_k(S^{k-1}),\Theta),\ \bar{\varphi}(U^0(\Theta),\Theta),\ V_k(S^{k-1}),U^0(\Theta))\}$, де передбачається, що $V_k(S^{k-1})$ обираються відповідно до стратегії π_l, математичне очікування обчислюється щодо випадкового процесу S^{k-1}, $\Phi(\bar{\varphi}(V_k(S^{k-1}),\Theta),\ \varphi(U^0(\Theta),\Theta),\ V_k(S^{k-1}),U^0(\Theta))$- деякий функціонал, що визначає поточні втрати перехідного процесу k- м такті.

Визначення по S^{k-1} призводить до очікуваних (середніх) втрат перехідного процесу в k- м такті. Таким чином $W_N(\pi_l)$– критерій якості перехідного процесу наближення від управління l- м такті (за неповною інформацією) до оптимального управління в умовах повної інформації на інтервалі управління довжини N для стратегії π_l.

Поставимо завдання синтезу рівномірно оптимальним (найкращим для кожного k, з усіх можливих стратегій у світлі отримуваної та оброблюваної

статистичної інформації) $\pi_l^{\text{опт}}$, яка здійснює переклад системи (у певному вище ймовірнісному сенсі) зі стану $\varphi\left(V_l\left(S^{l-1}\right),\Theta\right)$ у стан $\overline{\varphi}\left(U^0(\Theta),\Theta\right)$ при найменших очікуваних втрат перехідного процесу. Іншими словами, необхідно знайти таку стратегію управління $\pi_l^{\text{опт}}$, яка б задовольняла заданому критерію адаптації та доставляла б мінімум заданому критерію якості перехідного процесу на підінтервалі часу довжини N, починаючи з моменту l, тобто. найбільш повно використовувала б одержувану та оброблювану статистичну інформацію.

З літературних джерел відомо кілька підходів до вирішення цієї проблеми: міні-максний підхід; підхід Вальда; підхід Бернуллі у поєднанні з теоремою Байєса; фідуційний підхід; підхід максимальної правдоподібності.

Підходи міні-максної та максимальної правдоподібності засновані на припущенні, що будь-якому параметру в статистичному завданні рішення може бути приписаний деякий апріорний розподіл, відповідають так званому статистичному байєсовському підходу. Діяльність показано, що з досить загальних умов міні-максний підхід також є байесовским. Більш того, міні-максний підхід при деяких слабких обмеженнях є байєсовським щодо найменш сприятливого розподілу.

Проведені нами дослідження показали, що фідуціальний підхід та підхід максимальної правдоподібності не можуть призвести до оптимального вирішення проблеми. Можна отримати раціональний метод вибору (стосовно простоти обробки спостережень та якості перехідного процесу) апріорного розподілу для байєсовського підходу у разі використання його в задачах синтезу алгоритмів адаптивного управління.

У роботі пропонується новий підхід, що базується на модифікованій основі, сутність якого розглянемо на конкретному прикладі.

Нехай $f(S/\Theta)=\frac{1}{\Theta}exp\left\{-\frac{S}{\Theta}\right\}, 0<S<\infty$ (Щільність експоненційного розподілу), $\varphi(S_k,U_k)=(U_k-S_k)^2$,

$$Z_{k-1}=t_{k-1}=\sum_{i=1}^{k-1}S_i,\ \Phi\left(\varphi\left(\omega_k\left(\hat{\Theta},\alpha_k\right),\Theta\right),\overline{\varphi}\left(U^0(\Theta),\Theta\right),\omega_k\left(\hat{\Theta}_k,\alpha_k\right),U^0(\Theta)\right)=$$

$$=\overline{\varphi}\left(\omega_k\left(\hat{\Theta}_k,\alpha_k\right),\Theta\right).$$

Рішення:

1. Вибірковий розподіл $Z_{k-1}=t_{k-1}$:

$$dP\left(t_{k-1}/\Theta\right)=\frac{1}{T(k-1)\Theta^{k-1}}t_{k-1}^{k-2}\,exp\left\{-\frac{t_{k-1}}{\Theta}\right\}\alpha t_{k-1}, 0<t_{k-1}<\infty.$$

2. Фідуційний розподіл параметра Θ:

$$dP^*\left(\Theta/t_{k-1}\right)=\frac{t_{k-1}^{k-1}}{T(k-1)}\left(\frac{1}{\Theta}\right)^k exp\left\{-\frac{t_{k-1}}{\Theta}\right\}d\Theta, 0<\Theta<\infty.$$

3. Фідуційна оцінка густини розподілу ймовірностей S_k:

$$f^*(S_k/t_{k-1})=\frac{k-1}{t_{k-1}}\left(1+\frac{S_k}{t_{k-1}}\right)^{-k},\ 0<S_k<\infty.$$

4. Модифікована фідуціальна оцінка щільності розподілу ймовірностей S_k: $f^*(S_k/t_{k-1},\alpha_k)=\frac{k-1+\alpha_k}{t_{k-1}}\left(1+\frac{S_k}{t_{k-1}}\right)^{-k-\alpha_k},\ 0<S_k<\infty.$

5. Оптимальна функція $\omega_k^*(t_{k-1},\alpha_k)$ з умови:

знайти $\min\limits_{U_k}\int\limits_0^\infty (U_k-S)^2\frac{k-1+\alpha_k}{t_{k-1}}\left(1+\frac{S}{t_{k-1}}\right)^{-k-\alpha_k}dS$,

отримуємо $\omega_k^*(t_{k-1},\alpha_k)=\frac{t_{k-1}}{k-2+\alpha_k}$;

6. Оптимальний параметр α_k^0 з умови: знайти $\min\limits_{\alpha_k} M\limits_{t_{k-1}}\{\overline{\varphi}(\omega_k^*(t_{k-1},\alpha_k),\Theta)\}$,

де $\overline{\varphi}(\omega_k^*(t_{k-1},\alpha_k),\Theta)=\frac{t_{k-1}^2}{(k-2+\alpha_k)^2}-2\frac{t_{k-1}}{(k-2+\alpha_k)}\Theta+2\Theta^2$, отримуємо $\alpha_k^0=2$.

7. Оптимальне керування $U_k^0(t_{k-1})$ в k - м такті: $U_k^0(t_{k-1})=\frac{t_{k-1}}{k}$.

8. Оптимальна стратегія управління на базі інформативної статистики $z=t:\pi_l^{\text{опт}}(t)=\left\{\frac{t_{l-1}}{l},\dots\frac{t_{k-1}}{k},\dots,\frac{t_{N-1}}{N}\right\}$.

При цьому досягається досить суттєве покращення якості перехідного процесу наближення від управління в умовах неповної інформації до оптимального управління в умовах повної інформації.

Перевага запропонованого підходу у тому, що дозволяє також досить легко визначати рівномірно оптимальні стратегії управління за наявності обмежень на управління.

У цьому випадку використовуються оптимальні модифіковані оцінки фідуціальних щільностей $f^*(S_k/Z_{k-1},\alpha_k^0),k=\overline{l(1)N}$, при цьому з'являється можливість вказати раціональне (стосовно простоти обробки спостережень та якості перехідного процесу) апріорний розподіл для байєсовського підходу. У цьому усувається суб'єктивний момент під час виборів підходу.

Використання запропонованого модифікованого підходу сприятиме підвищенню якості алгоритмів адаптивного управління виробництвом і функціонування існуючих.

10.6. Спосіб адаптивної алгоритмізації завдань розрахунку виробничих програм для інформаційних систем

Основою економічного розрахунку промислових підприємств є планування. До найактуальніших завдань, у сфері планування, ставляться завдання розрахунку виробничих програм, дають оптимальні чи близькі до

оптимальним (за обраними критеріями) техніко-економічні і оперативні плани, багато з яких є цілими задачами з Булевими змінними.

Як правило, для вирішення завдань такого типу застосовуються методи відшукання точного рішення або наближені методи цілісного програмування, пов'язані з різними схемами впорядкованого перебору, які заздалегідь обумовлюються, тобто складаються згідно з деякими жорсткими програмами перебору. Як випливає з аналізу літературних джерел, для вирішення завдань такого типу найчастіше застосовується метод гілок та кордонів. Однак ефективність роботи алгоритму, заснованого на даному методі, суттєво залежить від того, наскільки вибір конкретного методу відображає специфіку задачі.

Згідно, в даний час накопичено досвід застосування імовірнісних методів для вирішення задач планування. Імовірнісний підхід до вирішення (нелокальних) завдань дозволяє багато алгоритмів інтерпретувати в рамках звичайної локальної схеми (типу градієнтної), правда вже для деякого наближеного (усередненого) завдання. Цей факт дає змогу вказати певний варіаційний підхід для формування ймовірнісних ітераційних алгоритмів розв'язання дискретних завдань.

Застосуємо описану вище схему імовірнісних алгоритмів до задачі лінійного програмування з змінними булевими $\sum_{j=1}^{n}\sum_{k=1}^{r_j} c_j^k x_{jk} \to \min\limits_{x},$ при обмеженнях:

$$\sum_{j=1}^{n}\sum_{k=1}^{r_j} a_{ij} x_{jk} \ge b_i, i = 1,2, ..., m,$$

$$\sum_{k=1}^{r_j} x_{jk} = 1, j = 1,2, ..., n,$$

$$0 \le x_{jk} \le 1; x \text{ - ціле}, j = 1,2, ..., n, \text{ k} = 1,2, ..., r_j .$$

Зв'яжемо з цим завданням відповідну функцію Лагранжа і перейдемо до ймовірнісної постановки завдання:

$$\Phi(x,\lambda) = \sum_j \sum_k c_j^k x_{jk} + \sum_i \lambda_i \left(b_i - \sum_j \sum_k a_{ij}^k x_{jk} \right) \to min,$$

де $x = \left[x^1, x^2, ..., x^n\right]$ - блочне представлення векторів, причому

$$x^j = \left(x_{j1} x_{j2}, .., x_{j_n r_j}\right), 0 \le x_{jk} \le 1; x \text{ - ціле}, k = 1, 2, ..., r_j, \qquad (22)$$

$$\sum_k x_{jk} = 1,\ j = 1,2,\ ...,\ n. \tag{23}$$

тобто. до завдання виду $M_x\Phi(x,\lambda) \to min$, де x – випадковий вектор, реалізації якого задовольняють вимогам (48), (49) і який має деяку щільність розподілу $P_x(\tau)$.

Задаватимемо рух у просторі випадкових векторів ітеративною формулою

$$x^N = \overline{U}^N x^{N-1} + U^n y^N, \tag{24}$$

де y^N - випадковий вектор, визначальний напрямок руху, що його реалізації задовольняють вимогам (23), (24), тобто. $y = \left[y^1, y^2, ..., y^n\right]$, $y^j = \left(y_{j1} y_{j2} y_{j3}, .., y_{n_j r_j}\right)$, $0 \le y_{jk} \le 1$, $k = 1,2,\ ...,\ r_j$, $\sum_k y_{jk} = 1$, із щільністю розподілу $P_y(S)$. U – Бульова випадкова величина така, що $P(U=1) = P_u$, $P(U=0) = q_u$.

Будемо, як і раніше, визначати напрямок руху, виконуючи правило локального поліпшення у вигляді $M_x^N\Phi(x,\lambda) - M_x^{N-1}\Phi(x,\lambda) < 0$.

Розпишемо докладніше $M_x^N\Phi(x,\lambda)$:

$$M_x^N\Phi(x,\lambda) = \int \Phi(t,\lambda)\delta(t - \overline{\mu}\tau - \mu s)P_u^N(\mu)P_x^{N-1}(\tau)P_y^N(S)d\mu d\tau ds dt =$$
$$= P_u^N \int \Phi(S,\lambda)P_y^N(S)dS + q_u^N \int \Phi(\tau,\lambda)P_x^{N-1}(\tau)d\tau.$$

Тоді різниця $M^N\Phi - M^{N-1}\Phi$ запишеться так:

$$M_x^N\Phi(x,\lambda) - M_x^{N-1}\Phi(x,\lambda) = P_u^N \int \left[\Phi(S,\lambda) - \Phi(\tau,\lambda)\right]P_y^N(S)P_x^{N-1}(\tau)d\tau ds.$$

$$P_U^N = M^N u,\ \int \Phi(S,\lambda)P_y^N(S)dS = M_y^N\Phi(y,\lambda),$$

Зауважимо, що $\int \Phi(\tau,\lambda)P_x^{N-1}(\tau)d\tau = M^{N-1}\Phi(x,\lambda)$, і, отже, через незалежність випадкових векторів у і x

$$M_x^N\Phi - M_x^{N-1}\Phi = M_u^N M_y^N M_x^{N-1}\left\{u\left[\Phi(y,\lambda) - \Phi(x,\lambda)\right]\right\} < 0.$$

При цьому

$$\Phi(y,\lambda) - \Phi(x,\lambda) = \sum_j \sum_k \left(c_j^k - \sum_i \lambda_i a_{ij}^k\right)\left(y_{jk} - x_{jk}\right).$$

Таким чином, завдання полягає у визначенні випадкового вектору y^N, такого, що

$$M_u^N M_y^N M_x^{N-1}\left\{u\sum_j\sum_k\left(c_j^k-\sum_i\lambda_i a_{ij}^k\right)\left(y_{jk}-x_{jk}\right)\right\}<0. \qquad (25)$$

Булева випадкова величина U першому рівні роботи алгоритму не змінюється, тобто. $P_u^N = P_u^{N-1}$,N = 1, 2, ...

Залежно від виконання операції опосередкування по випадковим змінним нерівності (26) тут, як і раніше, можна отримувати різні типи алгоритмів, що працюють як з реалізаціями випадкових векторів, так і з ймовірнісними характеристиками. Так, не проводячи операції зосередження, отримаємо алгоритм, що працює лише з реалізаціями випадкових векторів. При цьому можна визначити y^N так, щоб максимально гарантувати виконання нерівності (26). З цією метою припустимо, що

$$y_{jS}^N = 1\text{, якщо } c_j^S-\sum_i\lambda_i a_{ij}^S=\min_k\left[c_j^k-\sum_i\lambda_i a_{ij}^k\right],$$
$$y_{ji}^N = 0 \text{ для всіх i = 1,2,…, } r_j,\ i\neq S.$$

Рух (50) розуміється у разі як дію над реалізаціями випадкових векторів.

Виконуючи операцію опосередкування в нерівності (26) по Бульовій випадковій величині *U*, а також випадковому вектору у, отримаємо запис умови локального поліпшення у вигляді

$$M_x^{N-1}\left\{P_U^N\sum_j\sum_k\left(c_j^k-\sum_i\lambda_i a_{ij}^k\right)\left(\Pi_{jk}^N-x_{jk}\right)\right\}<0,$$

де $\Pi_{jk}=P\left(y_{jk}=1\right)$.

Визначимо змінні P_u^N і Π_{jk}^N наступним чином: $P_u^N = P_u^{N-1}$, N = 1,2, ..., $\Pi_{jS}^N = 1$, якщо $c_j^S-\sum_i\lambda_i a_{ij}^S=\min_k\left[c_j^k-\sum_i\lambda_i a_{ij}^k\right]$, $\Pi_{ji}^N = 0$,для всіх $i\neq S$.

Рух (25) здійснюватимемо у разі у ймовірнісних характеристиках. Позначимо через P_{jk} ймовірність того, що випадкова величина x_{jk} приймає значення, що дорівнює одиниці: $P_{ik}=P\left(x_{jk}=1\right)$. Тоді запишемо рух (25) запишемо у вигляді $P_{ik}^N = q_u^N P_{ik}^{N-1} + P_u^N \Pi_{ik}^N$, k = 1, 2 ..., r_j.

У нашому випадку

$$P_{iS}^N = q_u^N P_{iS}^{N-1} + P_u^N,\ P_{ji}^N = q_u^N P_{ik}^{N-1},\ i\neq S. \qquad (26)$$

Після проведення опосередкування за всіма випадковими змінними, присутніми в нерівності (26), маємо

$$P_u^N \sum_j \sum_k \left(c_j^k - \sum_i a_{ij}^k \lambda_i \right) \left(\Pi_{jk}^N - P_{ik}^{N-1} \right) < 0$$

$$P_u^N \sum_j \sum_k \left(c_j^k - \sum_i a_{ij}^k \lambda_i \right) \left(\Pi_{jk}^N - P_{ik}^{N-1} \right) < 0$$

Вважаючи $P_u^N = P_u^{N-1}$, $N = 1, 2\ldots$ можна визначити Π_{jk}^N з наступних умов:

$$\sum_{k=1}^{r_j} \Pi_{jk}^N = 1, \; j = 1,2,\ldots,n$$

$$P_{jS}^{N-1} < \Pi_{iS}^N \quad \forall S : c_j^S - \sum_{i=1}^m a_{ij}^S \lambda_i < 0,$$

$$P_{jS}^{N-1} > \Pi_{i\ell} \quad \forall \ell : c_j^\ell - \sum_{i=1}^m a_{ij}^\ell \lambda_i > 0,$$

при цьому рух здійснюється за (26).

Штучне залучення випадковості дозволяє формалізувати бажану схему перебору. При цьому відбувається своєрідне навчання перебору, пристосованого до специфіки цього завдання.

10.7. Адаптивна модель процесу стратегічного планування інформаційних систем із нестійкими умовами реалізації стратегії

Адаптивність стратегії, як властивість пристосування до умов її реалізації, визначається областю маневрування, під якою будемо розуміти безліч ресурсів, на основі яких робиться розрахунок і (або) коригування стратегії. Частина області маневрування співпадатиме з областю допустимих значень стратегії. Чим більша область маневрування R тим паче ефективним буде коригування стратегії у разі зміни умов її реалізації, і що більш вузька область маневрування, тим менше можливостей для коригування стратегії.

Оптимальна область маневрування визначається вирішенням наступного завдання стохастичного програмування:

$$R^* = Arg\,min \left\{ F(R) = M_\Theta \sum_{S=1}^{S} f_S(R,\Theta) \middle| R \geq \underline{R} \right\}, \qquad (28)$$

де M_Θ - Знак математичного очікування; $R^* = \{R_S^*\}, \quad S = \overline{1,S}$ - Вектор оптимальної області маневрування з урахуванням непрямого резерву; $R = \{R_S\}, \quad S = \overline{1,S}$ - Вектор обсягу ресурсів, необхідний для виконання обов'язкової частини

стратегії ІС $\underline{R}_S = \sum_{j=1}^{N} a_{S_j} \underline{X}_{j,s}$, $S = \overline{1,S}$, де $\underline{X}_j$ - Обов'язкова частина стратегії з виробництва продукції.

Вираз (28) $F(R)$ - Не диференційована функція, причому:

$$f_S(R,\Theta) = \sum_{S=1}^{S} max\{\overline{\alpha}_S(R_S - \Theta_S), \beta(\Theta_S - R_S)\},$$

де $\Theta = \{\Theta_S\}$, $S = \overline{1,S}$ - Випадковий вектор використання S-го ресурсу; $\overline{\alpha}_S$ – питомі витрати через надлишок S-го ресурсу; $\overline{\beta}_s$ – питомі витрати через дефіцит S-го ресурсу

Розв'язанням задачі (28) можна визначити і оптимальну область маневрування за виробничими потужностями, де $\overline{\alpha}_f$ – питомі витрати через простої f-ї групи обладнання, $\overline{\beta}_f$ – питомі витрати через дефіцит або наднормативні надлишки використання f-ї групи обладнання.

Оптимальна область маневрування повинна бути узгоджена з виробничою стратегією, тобто повинна розраховуватися на основі оптимальної області маневрування R^*. Однак розрахунок плану, що забезпечує максимально повне використання області R^* і випуск найприбутковішої продукції, є складне завдання узгодження. У кожному конкретному випадку питання узгодження оптимальної галузі маневрування зі стратегією випуску можуть вирішуватися по-різному, залежно від місії та цілей виробничо-економічної системи. Тому стратегія по-різному може бути узгоджена з оптимальною областю маневрування.

Збільшення адаптивних якостей стратегічних планів безпосередньо з удосконаленням системи критеріїв стратегічного планування і управління. У випадку завдання синтезу стратегії поведінки інформаційної системи, є багатокритеріальної завданням оптимізації. Дуже важливим є обґрунтування системи критеріїв ефективності та відповідних економічних показників, що відображають якість функціонування інформаційної системи, а також ефективність адаптації до зміни цільових установок плану (критеріїв оптимальності).

Стратегія, розрахована за одним критерієм оптимальності, має значно гірші маневрені якості, ніж багатокритеріальна стратегія. Доказ цього твердження очевидний і ґрунтується на законі необхідного розмаїття. Проблема багатокритеріальності при адаптивному стратегічному плануванні полягає в тому, що обране стратегічне рішення має бути інваріантним за зміни вектора переваги.

Аналіз зарубіжного досвіду та узагальнення вітчизняного досвіду функціонування інформаційної системи дозволяє виділити такі локальні стратегічні цілі: прибутковість, ринки, продуктивність, продукція, фінансові ресурси, виробничі потужності, дослідження та впровадження нововведень,

організація трудових ресурсів, соціальна відповідальність.

Для класу інформаційної системи з нестійкими умовами реалізації стратегії одним із найактуальніших критеріїв є критерій максимального використання оптимальної галузі маневрування плану. Чим більший вплив факторів, що дестабілізують, на систему, тим більшої актуальності набуває цей критерій у сукупності з іншими критеріями, наприклад максимумом прибутку. У свою чергу, чим більший рівень використання оптимальної області маневрування, тим більша глибина адаптивності стратегії.

Міра глибини адаптивності стратегії повинна відображати роздільний вплив кожної характеристики області маневрування: маневрування на основі зміни допустимого значення стратегії; маневрування з урахуванням зміни цільових установок плану.

Для цього введемо поняття загальної глибини адаптивності стратегії, що визначається парою:

$$H = \left\langle H^*, \tilde{H} \right\rangle, \tag{29}$$

де H - загальна глибина адаптивності стратегії; $\tilde{H}$ - Глибина адаптивності по області допустимого значення стратегії; H^* – глибина адаптивності щодо зміни цільових установок стратегії.

У виразі (29) $\tilde{H}$ визначається виразом:

$$\tilde{H} = \prod_{S=1}^{S} H_S, \quad H_S = min\left\{ \frac{R_S^{**}}{R_S^*}, 1 \right\}, \quad S = \overline{1,S} \tag{30}$$

де R_S^{**}, R_S^* – оптимальна область маневрування відповідно до та після погодження зі стратегією ІС.

При зміні глибини адаптації, при зміні цільових установок стратегії передбачається, що існує більше однієї цільової функції і існує можливість зміни вектора переваг у межах цих функцій у відомих межах.

Нехай вирішується багатокритріальне завдання пошуку точки розв'язання X^* з області Ω. λ – сукупність векторів переваг даних критеріїв f_i, $i = \overline{1,m}$ m – метричний простір $\left(\lambda \in \wedge / \sum_{i=1}^{m} \lambda_i = 1, \lambda_i \geq 0 \right)$.

Позначимо R^* наступним чином: $\lambda \subset R^*$, причому: $R^* \subset R^+$, $r \in R^* : \underline{r}_i \leq r_i \leq \bar{r}_i$, $i = \overline{1,m}$, де $\underline{r}_i, \bar{r}_i$ – відповідно мінімально та максимально допустимі значення компонентів вектора переваг, причому може виконуватися умова:

$$\sum_{i=1}^{m} r_i > 1 \tag{31}$$

Знак «=» (31) означав би наявність одного вектора λ зі значенням компонент рівних $1/m$. У загальному випадку точки $\lambda_i \subset \wedge$ віддалені в середньому на однакові відстані, як від точки $\underline{r}$, так і від точки $\bar{r}$.

Здатність інформаційної системи до реагування на зміни λ означає, що обране рішення X^* стратегії має породжувати в області R^* такий вектор λ відносні відстані від якого до точок $\underline{r}, \bar{r}$ мінімальні та однакові (майже однакові). Це забезпечить найбільше зближення у просторі Ω крапки X^* з точками, що відповідають допустимим вектором переваг.

Таким чином, глибина адаптації H^* стратегії, обраної з урахуванням багатокритеріальності на заданій галузі $\wedge$, Розраховується за формулою:

$$H^* = \underline{H} / \overline{H} \le 1, \tag{32}$$

де

$$\underline{H} = \frac{1}{m}\sum_{i=1}^{m} \underline{r}_i H_i^*, \tag{33}$$

$$\overline{H} = \frac{1}{m}\sum_{i=1}^{m} \bar{r}_i H_i^*. \tag{34}$$

У виразах (28) – (34) H_i^* - індивідуальні критеріальні оцінки глибини адаптації стратегії, причому:

$$H = \begin{cases} 1 - \dfrac{f_i^* - f_i^{**}}{f_i^{**} - f_i^*}, i \in I_1 \\ 1 - \dfrac{f_i^{**} - f_i^*}{f_i^* - f_i^{**}}, i \in I_2 \end{cases}, \tag{35}$$

де I_1, I_2 - безліч функцій, які максимізуються та мінімізуються, відповідно, $f_I = \max\limits_{x \in \Omega} f_i(X), \quad i = \overline{1,m}$.

Частина i-го критерію H_i^* при розрахунку глибини адаптації визначається за такою формулою:

$$H_i^* = \frac{\underline{r}_i H_i}{\sum\limits_{i=1}^{m} \bar{r}_i H_i^*}. \tag{36}$$

Таким чином (32) – (35) видно, що впливати на глибину адаптації можна через керовані параметри $\underline{r}_i, \bar{r}_i, H_i^*$.

Індивідуальні оцінки H_i^* при цьому визначаються заданим вектором переваги $\lambda_i, i = \overline{1,m}$. що породжує точку вирішення багатокритеріальної задачі.

З виразів (32) – (35) видно, що глибину адаптації можна збільшити з допомогою звуження області $\wedge$ що призводить до зближення граничних точок. Якщо область $\wedge$ задано жорстко, глибина адаптації залежатиме від оцінок H_i^*, Зміна яких можливо здійснити завданням нового вектору переваги.

При реалізації алгоритму вирішення багатокритеріальної задачі збільшення глибини адаптації плану досягається на кожному кроці шляхом надання більшого рівня пріоритету тим критеріям, часткова участь яких у H_i^* мінімально. При цьому вирішується завдання: $min\underline{r}_i H_i^*$, $i=\overline{1,m}$.

З умов (35) видно, що $H_i^* \in [0,1]$, причому чим ближче значення H^* до одиниці, тим паче адаптивні якості плану. Таким чином, при заданому діапазоні допустимих змін вектору переваг, пошук рішення, що має адаптивні якості, повинен супроводжуватися максимізацією значення H_i^*.

Існуючі алгоритми вирішення багатокритеріальних завдань по-різному сприяють виконанню вимог адаптації обраної стратегії щодо умов завдання переваг на безлічі цільових функцій. У цьому випадку найбільш перспективними є ті з них, які ґрунтуються на інтерактивних процедурах, за якими на кожному кроці виконання особа, яка приймає рішення (ЛПР), може змінити вектор переваг. Алгоритм, в основі якого закладена ідея звуження області парето-оптимальних рішень, дозволяє на кожному кроці заглибитися в цю область, виконуючи при цьому наведені вище вимоги адаптації.

У процесі реалізації стратегії дуже важливим аспектом є постійний моніторинг відхилень фактичних результатів функціонування від стратегічних цілей, з урахуванням яких виконується процес прийняття рішень коригування інформаційної системи. На практиці таке коригування найкраще проводити, розподіляючи стратегію на менші планові періоди.

Отже, коригування стратегії фактично виконується лише на рівні тактичного планування виробництва. У зв'язку з тим, що процес розподілу, згідно з логікою системного підходу, має безперервний характер, коригування стратегії має також безперервний характер.

Під час коригування стратегії виробничо-економічної системи виконується реалізація закладених у стратегії адаптивних якостей. У цьому відбувається розподіл області маневрування стратегії за плановими періодами, що, природно, висловлює специфіку математичної постановки завдання розподілу. Т.о., оптимальну адаптивну стратегію X^* необхідно розподілити на менші періоди, тобто здійснити перехід від стратегічного планування до тактичного планування та управління ІС.

Нехай $X^*(t)$, $t=\overline{0,T}$ - стратегія ІС, розподілена на періоди $[t,t+1]$ яка може бути отримана різними способами, наприклад, за допомогою рішення наступного завдання:

$$\sum_{t=0}^{T-1} X(t) \to max, \qquad (37)$$

$$X(t) \in X\ , \tag{38}$$

де X – допустима сфера визначення стратегії $X(t)$.

З розв'язання задачі (37) – (38) випливає визначення оптимальної області маневрування в період $[0,t]$:

$$R^*(t) = \sum_{k=0}^{t} R\left(X^*(k)\right), \quad R(0) = 0, \quad t = \overline{0,T}\ , \tag{39}$$

де $R^*(t)$-оптимальна область маневрування на період; $R\left(X^*(k)\right) = \left\{R_S\left(X^*(k)\right), S = \overline{1,S}\right\}$, причому $R_S\left(X^*(k)\right) = A_S X^*(k), S = \overline{1,S}, r = \overline{0,t}, t = \overline{0,T}$.

Дотримуючись термінології теорії оптимального управління обсяг ресурсів, що використовуються протягом періоду $[0,t]$, $R(t) = \{R_S(t)\}, \quad S = \overline{1,S}$ - стан об'єкта управління на період часу $[0,t]$. Величина $R^*(t)$ буде заданою (детермінованою) траєкторією поведінки об'єкта управління у період $[0,t]$.

Під $X(t), \quad t = \overline{0,T}$ розумітимемо в даному випадку розподілену стратегічну програму на період $(t,t+1)$, тобто під $X(t)$ розуміється управління на період $(t,t+1)$.

Таким чином, маємо ІС, поведінка якої може бути описана наступною системою рівнянь:

$$R(t+1) = R(t) + BX(t) + \Theta(t), \quad R(0) = 0, t = \overline{1,T}, \quad X(0) = \Theta(0) = 0 \tag{40}$$

План $X(t)$ належить допустимій області X :

$$B_f X_j(t) \le A(t), \quad t = \overline{0,T-1}, \tag{41}$$

$$\underline{X}(t) \le X(t), \quad t = \overline{0,T-1}, \tag{42}$$

$$X^* = \sum_{t=0}^{T-1} X(t). \tag{43}$$

Як критерій оптимальності в задачі розподілу стратегії за плановими періодами з урахуванням ймовірності природи функціонування ІС доцільно взяти мінімум очікуваного відхилення від оптимальної адаптивної стратегічної траєкторії поведінки системи $R^*(t), \quad t = \overline{0,T}$:

$$\min F\left(X(0),...,X(T-1)\right) = M_f\left(R(0),...,R(T),X(0),...,X(T-1),\Theta(0),...,\Theta(T-1)\right) \tag{44}$$

де $f(R(t),X(t),\Theta(t))=\max\limits_{0\le t\le T}\|R(t)-R^*(t)\|$; M - Знак математичного очікування; $\|\ \|$ - евклідова норма, яка показує максимальне за модулем відхилення координат.

У загальному вигляді модель завдання розподілу виробничої стратегії за плановими періодами, враховуючи (37) – (43), має вигляд:

$$MinF(X(0),\dots,X(T-1)),\ R(t+1)=R(t)+Bx(t)+\Theta(t),R(0)=0,t=\overline{0,T},X(t)\in X\ .$$

10.8. Методологія реалізації адаптивного автоматизованого пошуку оптимальних альтернатив прийнятих рішень

Для сучасного рівня розвитку систем управління інформаційної системи характерним є те, що більшість завдань управління та планування вирішується в умовах невизначеності, неточності та недостатності інформації про процеси та умови функціонування виробництва, про вплив зовнішнього середовища. При розробці моделей та алгоритмів, інформаційних технологій необхідно використовувати всі ці фактори та розглядати роботу виробничих систем в умовах невизначеності.

Існуючий рівень розвитку систем прийняття рішень (СПР) за умов невизначеності не задовольняє вимогам виробничих систем щодо адаптивності, ефективності, інформативності, адекватності. Це пов'язано насамперед з обмеженими можливостями алгоритмічного підходу. Процеси управління виробничими системами за умов невизначеності не піддаються апріорної алгоритмізації. Підвищити показники якості систем прийняття рішень за умов невизначеності можна шляхом використання підходів, в основі яких лежать логічні моделі підтримки прийняття рішень, що базуються на апараті нечітких множин.

Попередній аналіз цієї проблеми показує, що, по-перше, методи прийняття рішень багато в чому визначаються специфікою завдань та способом їхньої формалізації; по-друге, відсутні дослідження, пов'язані з використанням нечіткого вибору альтернатив за умов невизначеності. Більшість робіт присвячені методам, які не враховують невизначеність, що виникають у процесі управління виробничими системами.

Тому з метою підвищення ефективності та якості підтримки прийняття рішень в умовах невизначеності наші дослідження були спрямовані на розробку адаптивних методів та моделей прийняття рішень.

Досягнення поставленої мети у роботі вирішувалися такі:

- формалізація математичної моделі прийняття рішень;
- розробка системи прийняття рішень в управлінні виробництвом (СПРУВ) для реалізації СПР з використанням математичного апарату нечітких множин.

Наукова новизна запропонованих рішень полягає у використанні, для реалізації окремих етапів процесу прийняття рішень в умовах невизначеності, математичного апарату на основі нечітких множин, що дозволяє вирішувати

широке коло завдань щодо підвищення діяльності підприємств на етапі їх подальшого розвитку.

У випадку прийняття рішення можна визначити як перетворення інформації стану в кількісні чи якісні складові інформації управління. Якість процесу прийняття рішень перебуває у прямої залежності від повноти обліку всіх чинників, важливих для наслідків прийнятих рішень. Часто ці фактори мають суто суб'єктивний характер, властиві як особі, яка приймає рішення (ЛПР), так і будь-якій системі прийняття рішень. ЛПР часто вимушено діяти в умовах невизначеності, тобто в умовах відсутності необхідної кількості інформації для цілеспрямованої організації його дій у процесі прийняття рішень.

Існують різні види невизначеності. Не претендуючи на повноту, можна вказати на такі, що найчастіше зустрічаються:

- невизначеність, викликана нестачею інформації, її достовірності з технічних, соціальних та інших причин;
- невизначеність підходів ЛПР, через брак його досвіду та знання факторів, що впливають на прийняття рішень;
- невизначеність, пов'язана з обмеженнями у ситуаціях прийняття рішень (обмеження за часом та елементами простору параметрів, що характеризують фактори прийняття рішень);
- невизначеність, викликана зверненням середовища чи ворога, що впливає процес прийняття рішення.

Часткове або повне зняття невизначеності може бути досягнуто за рахунок наявної або додатково одержуваної інформації.

На підставі застосування теорії дослідження операцій, системного аналізу та теорії прийняття рішень нами пропонується наступна методика формування складу та послідовності 11-ти етапів процесу прийняття рішень в умовах невизначеності:

1. Отримання інформації про проблемну ситуацію стану об'єкта управління.
2. Формулювання проблемної ситуації (поточний її аналіз, з'ясування особливостей проблеми, що виникла, ступінь терміновості та необхідності її вирішення).
3. Визначення цілей вирішення проблеми та вибір критеріїв ефективності.
4. Виявлення та формування обмежувальних умов.
5. Розробка альтернатив, визначення можливих варіантів вирішення та попередній їх аналіз з метою вибору найбільш ефективних.
6. Формулювання постановки задачі.
7. Розробка логіко-математичної моделі завдання, що дозволяє оцінювати ефективність кожної альтернативи.
8. Аналіз та вибір методів розв'язання задачі, розробка алгоритму.
9. Оцінка альтернатив та визначення з них найбільш ефективною.
10. Прийняття рішень ЛПР на основі наявного досвіду з урахуванням

соціального, психологічного, технологічного та інших аспектів управління.

11. Реалізація та контроль прийнятих рішень.

Процес прийняття рішень за умов невизначеності є складною циклічною процедурою. Результат практично кожного з етапів може суттєво вплинути на постановку задачі та викликати її зміну. Формально процедура прийняття рішень у завданнях управління є деяким функціональним відображенням, яке формує рішення при початковому наборі даних у процесі пошуку.

Вважатимемо, що вихідна H і оцінювана G функції задані. Процедура вибору з безлічі можливих варіантів рішення полягає в використанні оцінюваної функції до вхідний. Визначимо вихідну функцію як відображення виду $H : L \times \Omega \to Y$, де L, Y і Ω - множини відповідно варіантів рішень, можливих результатів на виході та заходи невизначеності.

Тоді функцію, що оцінюється G визначимо як відображення виду $G : L \times Y \to E$, де E - безліч величин, зумовлених показниками.

Нехай X і X° - множини відповідно можливих і допустимих рішень, g - функція, що відображає $X \times \Omega$ в деяку кількість U, частково чи цілком упорядковано ставленням $\leq$, а τ - функція, що перекладає Ω в Y (функція допустимості), тобто $g : X \times \Omega \to E$, $\tau : \Omega \to E$, Тоді задачу знаходження задовільних рішень сформулюємо в такий спосіб: для заданої множини $X^{\circ} \leq X$ визначити також $x \in X$, щоб для всіх $\omega \in \Omega$ виконувався критерій задоволення виду $g(x,\omega) \leq \tau(\omega)$.

З допомогою нерівностей цього виду робимо відбір всіх варіантів, які задовольняють вимоги, прийнятому цьому етапі критерію. Загалом завдання визначення обґрунтованих рішень визначимо як сукупність значень $\left(g, \tau, X^{\circ}, \Omega\right)$, а рішенням у цьому випадку є $x \in X$, для якого виконується зазначена нерівність при $\omega \in \Omega$. Функцію мети g задаємо вихідної та оцінюваної функціями відповідно до виду:

$$H : X \times \Omega \to Y$$
$$G : X \times \Omega \times Y \to Y,$$

де Ω - безліч можливих факторів, які впливають на результат рішень X.

Тоді $g(x,\omega) = G\{x, \omega, H\}$.

Рішення X - вважатимемо задовільним, якщо воно призводить до значення оцінюваної функції, яка не перевищує певного рівня $\tau(\omega)$ за будь-яких $\omega \in \Omega$. Крім усього, якщо $\Omega = 0$, а це відповідає нагоді видимого зв'язку між L і Y у відсутності невизначеності) завдання можна привести до вигляду $g(x) = G\{xH/x\} \to min$.

Однією з основних вимог до засобів прийняття рішень є адаптація принципів управління, що реалізуються за умов функціонування системи управління. У динамічному середовищі з частковою невизначеністю ефективність прийнятих рішень залежить від визначеності та достовірності знань про стан об'єктів управління за умов функціонування системи

управління.

На підставі викладеного сформулюємо завдання процесу управління: на основі інформації, що отримується, з'ясувати стан керованих об'єктів та умов їх функціонування, визначити необхідні дії для управління.

Основним фактором, який визначає процес прийняття рішень, є ситуація S яка вимагає прийняття рішень щодо дії на керований об'єкт керуючої дії X та пошуку рішення $\varphi : X = \varphi(S)$.

Представимо модель автоматизованого пошуку рішень у вигляді кортежу:

$$\sum = \langle T, I, W, D_I, D_X, Z, V, P, X, Q, \Theta, \varphi \rangle,$$

де T - безліч моментів часу;

I - Інформація про стан об'єкта управління;

W - інформація про стан довкілля;

D_I - множина допустимих значень стану об'єкта управління;

D_X - безліч допустимих управляючих процесів;

Z - безлічі цілей управління;

V - Можливості обчислювальної техніки;

P - Відомості про систему пріоритетів;

X - множин управляючих процесів;

Q - безліч зв'язків між I і X ;

Θ - безліч закономірностей діяльності об'єкта;

φ - Відображення, що характеризує процес пошуку рішень.

Для реалізації відображення $\varphi : I \to X$ пропонується наступна послідовність логіко-математичних етапів процесу прийняття рішень:

1. Формулювання проблемної ситуації S:

$$\varphi_1 = \langle T, I, D_I, W, Z, S \rangle;$$

2. Класифікація ситуацій:

$\varphi_2 = \langle S, K_1, K_2, P_1 \rangle$,

де K_1, K_2 і P_1 - безліч відповідно класів ситуацій, правил класифікації та експертних переваг під час оцінки ситуації $(P_1 \in P)$;

3. Вибір стратегії пошуку рішень (формування завдань)

$$\varphi_3 = \langle S, Q, \Theta, R, P_2 C \rangle,$$

де R - Ресурси для ліквідації проблемної ситуації; P_2, C - безлічі відповідно переваг під час вибору стратегій та стратегій пошуку керуючих рішень (корегування виробничих планів, заміна ресурсів).

4. Побудова моделі пошуку рішення

$$\varphi_4 = \langle S, K, C, P_3, M \rangle,$$

де P_3, M - множини відповідно переваг ЛПР під час моделювання та формування моделей.

5. Конструювання процедури пошуку рішень

$$\varphi_5 = \langle M, V, P_4, A \rangle,$$

де V, P_4, A - множини відповідно до можливостей засобів обчислювальної системи (моделі, алгоритми, модулі, засоби спілкування тощо), переваги ЛПР під час конструювання процедури пошуку рішень та алгоритмічних процедур пошуку рішень.

6. Формування варіанта вирішення

$$\varphi_6 = \langle M, A, X, F, P_5 \rangle,$$

де F - безліч критеріїв оцінки корисності рішення.

7. Вибір рішення

$$\varphi_7 = \langle M, F, P_6, X^* \rangle,$$

де P_6 - безліч переваг (неформального характеру) під час вибору рішення; X^* - найкраще рішення, прийняте ЛПР з урахуванням оцінок.

Перелічені етапи процесу пошуку рішень дають можливість сформулювати постановку задач адаптації, що складається у визначенні процедури пошуку A керуючих дій X відповідно до стану об'єкта I та впливу зовнішнього середовища W.

Процес пошуку альтернатив складається із трьох основних функціональних блоків:

Блок 1 – формування ситуацій S, призначений для виявлення та опису певною мовою ситуації S на основі аналізу інформації про стан об'єкта управління I та зовнішнього середовища W, досвіду ЛПР, функціонування цього блоку, що включає етапи φ_1 і φ_2 визначимо наступним відображенням

$$\psi_1 : I \times W \times P \to S.$$

Блок 2 – конструювання моделі M - призначений для створення логіко-математичних моделей пошуку альтернатив для вирішення відповідної поточної проблемної ситуації; функціонування цього блоку, що містить етапи φ_3 φ_4, визначимо наступним відображенням:

$$\psi_2 : S \times C \times P \to M .$$

Блок 3 – визначення процедур пошуку A - призначений для формування та вибору альтернативних дій на основі моделі M, процедури пошуку альтернатив рішення A та системи переваг ЛПР. Функціонування цього блоку, що містить етапи φ_5, φ_6 і φ_7 визначимо наступним відображенням:

$$\psi_3 : M \times A \times P \to X .$$

Загалом адаптивна модель пошуку альтернатив рішення має вигляд:

$$I \to S \to M \to X ,$$
$$\uparrow Y_1 \uparrow Y_2 \uparrow Y_3 .$$

де Y_1, Y_2, Y_3 - функції відповідно до опису ситуації деякою мовою, перекладу його у формалізовану модель та визначення процедури формування рішення.

Наведені етапи функціонування мають здатність адаптуватися до ситуації за рахунок конструювання та коригування елементів процесу пошуку альтернатив рішень (моделей, стратегій, алгоритмів).

Процес формалізації завдань прийняття рішень у конкретній предметній області може бути представлений у вигляді моделі, вираженої сукупністю множин

$$M = \langle X, A, Y, F \rangle ,$$

де X, A, Y - безлічі відповідно керуючих, фіксованих та керованих параметрів середовища; F - безліч функціональних залежностей, які пов'язують елементи множин X, A, Y . Для кращої реалізації діалогу, у багатьох X, A, Y введемо короткі словесні характеристики відповідних параметрів. У цьому випадку безлічі X і Y будуть представлені як набори впорядкованої пари виду:

$$X = \left\{ \left(h_i, x_i \right), i \in I = \overline{1,n} \right\}; Y = \left\{ \left(h_i, Y_i \right), j \in J = \overline{1,m} \right\},$$

а множини A - як набір упорядкованих трійок виду $C = \left\{ \left(h_\ell, a_\ell, a_\ell^R \right), \ell \in L = \overline{1,L} \right\}$,

де h_i, h_j, h_ℓ - Назву відповідних параметрів; x_i, y_j, a_ℓ - їх умовні позначення; a_ℓ^R - Відомі кількісні значення фіксованих параметрів.

Модель цього виду відображає основні закономірності конкретного середовища та залежності виду $Y_j = f\left(x_1, x_2, ..., x_n, a_1, ..., a_n \right)$, які їх пов'язують З іншого боку, модель дозволяє формувати в діалоговому режимі різні постановки завдань прийняття рішень певного класу.

Порівняльна оцінка якості СПР, розроблених із застосуванням СПРУВ та з використанням традиційних підходів показала, що застосування СПРУВ дозволяє покращити якість та ефективність СПР в управлінні ІС, тому може

бути рекомендована як допоміжний компонент, який складається із засобів моделювання прийняття рішень, та контролює процес функціонування СПР.

Контрольні питання до розділу 10

1. Перерахуйте основні етапи процесу побудови математичної моделі.
2. Дайте визначення концептуальної та математичної постановки задачі.
3. З якою метою застосовується перевірка адекватності моделі?
4. Опишіть два принципи побудови моделі.
5. Які підходи до побудови математичної моделі ви знаєте? У чому вони?
6. Сформулюйте складові похибки під час використання чисельних методів.
7. Дайте визначення коректності математичної моделі.
8. Перерахуйте основні етапи циклу обчислювального експерименту.

СПИСОК ЛІТЕРАТУРИ

1. Ismailov K.Y. Training of police officers to search and analize significant information from open sources (example of chat-bott applications). Sciences of Europe (Praha, Czech Republic) VOL 5, No 48 (2020). P. 17-25.
2. Ismailov K.Y., Nebeska M.S. The threat of autarky in conditions of electoral information sovereignty. European Reforms Bulletin. 2018. № 4. P. 13-19.
3. Ismailov, K. (2020). To the issue of personal information circulation in the National police databases. Fundamental and applied researches in practice of leading scientific schools, 38 (2). Canada. P. 41-45.
4. Ismailov K.Y. Peculiarities of human rights and freedom while applying intelligence-led policing (ILP). Scientific Bulletin of the Dnipropetrovsk State University of Internal Affairs. 2019. Special Issue № 1. P. 36-41.
5. Zaiets O.M. Application software IBM I2 ANALYST'S NOTEBOOK in law enforcement Ukraine for pretrial investigation of criminal offenses. European Reforms Bulletin. 2016. № 1. P. 69-72.
6. Zaiets Oleksandr, Ismailov Karen and others. Information and analytical support of public administration in the field of insurance. Public administration in the digital economy: monograph. Tallinn. Scientific Center of Innovative Researches OÜ. 2020. 160 p. (P. 91-105). URL: https://mono.scnchub.com/index.php/about/catalog/view/3/32/180-1
7. Сучасні інформаційні системи і технології: управління знаннями : навч. посібник / В. М. Антоненко, С. Д. Мамченко, Ю. В. Рогушина. Ірпінь : Нац. університет ДПС України, 2016. 212 с.
8. Балтовський О.А., Ісмайлов К.Ю., Сіфоров О.І., Форос Г.В., Заєць О.М. Теорія систем і системний аналіз: навч. посібник. Одеський держ. унів-т внутр. справ, 2020. 156 с.
9. Бахрушин В.Є. Методи аналізу даних: навчальний посібник для студентів. Запоріжжя : КПУ, 2011. 268 с.

10. Бурячок В. Л. Інформаційний та кіберпростори: проблеми безпеки, методи та засоби боротьби: підручник. К. : ТОВ «СІК ГРУП Україна», 2015. 449 с.
11. Бурячок В.Л. Основи формування державної системи кібернетичної безпеки: монографія. К.:НАУ, 2013. 432 с.
12. Бутенко Т.А. Інформаційні системи і технології в управлінні організацією: Технологія створення реляційних баз даних: метод. вказівки та завдання для самостійної роботи для студентів ОКР «спеціаліст» і «магістр» денної та заочної форми навчання галузі знань 0306 «Менеджмент і адміністрування». Х., 2017. 27с.
13. Воронін А. М. Інформаційні системи прийняття рішень: навчальний посібник. / Воронін А. М., Зіатдінов Ю. К., Климова А. С. К. : НАУ друк, 2009. 136с.
14. Інформаційні технології: підруч. / В.Б. Вишня, К.Ю. Ісмайлов, І.В. Краснобрижий, С.О. Прокопов, Е.В. Рижков. Дніпро : ДДУВС, 2020. 425 с.
15. Ковальчук В. В. Основи наукового дослідження: навч. посібник / В. В. Ковальчук, Л. М. Моїсеев. К. : Видавничий дім „Професіонал", 2008. 240 с.
16. Лебідь М. Т. Багаторівневі схеми реалізації загальногалузевих задач / М.Т. Лебідь, С.І. Шулудько / Харк. нац. аграр. ун-т ім. В.В.Докучаева. Харків, 2008.
17. Метешкін К. О., Костенко О. Б., Сенчук Т. С.. Інформаційні системи і технології. Х., 2010. 240 с.
18. Інформаційні системи. Навч. посібн. /за наук. ред. Н. В. Морзе; Морзе Н.В., Піх О.З. Івано-Франківськ, «ЛілеяНВ», 2015. 384 с.
19. Дьоміна В.М. Оптимізаційні методи та моделі. Лінійне програмування: конспект лекцій. Х.: ХНАУ, 2015. 75 с.
20. Дьоміна В.М. Оптимізаційні методи та моделі. Моделювання систем масового обслуговування: конспект лекцій. Х.: ХНАУ, 2015. 42 с.
21. Основи кримінального аналізу: підручник / Бабенко А.М., Заєць О.М., Некрасов В.А., Ісмайлов К.Ю., Пефтієв Д.О. та ін.; за заг. ред. Користіна О.Є., 2019. 296 с.
22. Павлиш В. А., Гліненко Л. К. Основи інформаційних технологій і систем: Навчальний посібник. Львів: Видавництво Львівської політехніки, 2017. 500 с.
23. Поморцева О. Є. Лабораторний практикум з навчальної дисципліни "Комп'ютерні засоби в економіці та підприємництві": навчальнопрактичний посібник / О. Є. Поморцева ; Харк. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. Х. : ХНУМГ ім. О. М. Бекетова, 2017. 127 с.
24. Правова інформація та комп'ютерні технології в юридичній діяльності: навч. посіб. / В.Г. Іванов, С.М. Іванов, В.В. Карасюк та ін.; За заг. ред. В.Г. Іванова. Х.: Право, 2010. 240 с.

25. Сендзюк М.А. Інформаційні системи і технології в економіці: навч.-метод. посіб. для самост. вивч. дисципліни / М.А. Сендзюк; М-во освіти і науки України, ДВНЗ “Київ. нац. екон. ун-т ім. В. Гетьмана”. К. : КНЕУ, 2018. 68 с.
26. Синявіна Ю.В. Математичне програмування: навч.-метод. посібник / Ю.В. Синявіна, М.Т. Лебідь / Харк. держ. аграр. ун-т ім. В.В. Докучаєва. Х., 2007. 72 с.
27. Синявіна Ю.В. Математичне програмування: навч.-метод. Посібник. Х., 2017. 72 с.
28. Соколов В.Ю. Інформаційні системи і технології : Навч. посіб. К. : ДУІКТ, 2018. 138 с.
29. Сучасні підходи щодо адаптивного автоматизованого управління складними системами в умовах невизначеності: монографія / Кокошко В.С., Балтовський О.А., Ісмайлов К.Ю., Сіфоров О.І., Форос Г.В. та ін. Одеса: ОДУВС, 2019. 378 с.
30. Тактичний кримінальний аналіз: теорія та практика; навчальний посібник / О.Є. Користін, Н.П. Свиридюк, О.М. Цільмак, О.М. Заєць, К.Ю. Ісмайлов, В.А. Некрасов; МВС України, ДНДІ, ОДУВС. Одеса: РВВ ОДУВС, 2019. 216 с.
31. Телекомунікаційні та інформаційні мережі: підручник / П.П. Воробієнко, Л.А. Нікітюк, П.І. Резніченко. К.: САММІТ-Книга, 2010. 708 с.: іл.
32. Ульянченко О. В. Методи оптимізацій в економіці: Навч. посібник Харків, 2019. 139 с.
33. Ульянченко О.В. Дослідження операцій в економіці: підручник. Суми: Видавництво «Довкілля», 2018. 594 с. (з грифом Міністерства освіти і науки України).
34. Ульянченко О.В. Математичне програмування: Навч.посібник / О.В. Ульянченко, М.Т. Лебідь, Г.Г. Хлівняк, В.О. Бабенко. Харків, 2002. 296 с.
35. Ушакова І. О. Основи системного аналізу об’єктів та процесів комп’ютеризації : навчальний посібник. Ч. 2 / І. О. Ушакова. Х. : Вид. ХНЕУ, 2008. 324 с.
36. Ушакова І. О. Практикум з навчальної дисципліни "Основи системного аналізу об’єктів і процесів комп’ютеризації": навчальний посібник / І. О. Ушакова, Г. О. Плеханова. Х. : Вид. ХНЕУ, 2018.– 344 с.
37. Шулудько С.І. Інформатика та обчислювальна техніка: Навч. Посібник / С.І. Шулудько, О.І. Нестеренко, Н.М. Проценко / Харк. держ. агр. ун т ім. В.В.Докучаева 2017. 320 с. (з грифом Міністерства освіти і науки України)
38. Шулудько С.І. Інформатика та обчислювальна техніка: Навч. Посібник / С.І. Шулудько, О.І. Нестеренко, Н.М. Проценко. Харків, 2015. 320
39. Щербаков П.А. Інформатика та комп'ютерна техніка: програмне забезпечення ЕОМ для студентів екон. спец. / П.А. Щербаков, О.В.

Ульянченко, Н.М. Мартьянова, Т.А. Бутенко / Харк. держ. аграр. ун-т ім. В.В. Докучаева. Харків, 2019. 292 с. (з грифом Міністерства аграрної політики України)

40. Щербаков П.А., Ульянченко О.В. Інформаційні системи та технології в аграрному менеджменті: теоретичні й організаційні основи: Навч. посібник. Харків, 2020. 162 с.