**Лекція №19. Взаємодія, еволюція програм і даних**

Готові програмні системи (далі – ПС) активно використовують при створенні і супроводі нових  систем. При цьому виникають різного роду помилки, які вимагають внесення змін у систему після того, як помилка виявлена або виникла необхідність у зміні або покращенні певних характеристик системи.

На відміну від технічного забезпечення, яке з часом вимагає ремонту, програмне забезпечення не «зношується», і тому процес супроводу націлено насамперед на еволюцію системи, тобто не тільки на виправлення помилок, а й на заміну її окремих функцій і можливостей.

Типові причини внесення змін це:

–  виявлення дефектів в системі під час експлуатації, які не були виявлені на процесі тестування;

–  з'ясування невідповідності або невиконання деяких вимог замовника, через що система не виконує окремі функції;

– зміна умов замовником, які пов'язані з коригуванням раніше поставлених їм вимог.

Як стверджують експерти, процес внесення змін в експлуатовану систему досить дорогий, його вартість досягає від 60 до 80 %  загальної вартості розробки системи.

До видів супроводу належать**:**

– коригування – внесення змін в діючу  ПС для усунення помилок, які були виявлені після передачі системи до експлуатації;

– адаптація продукту до змінених умов (апаратури, ОС) використання системи після її передачі в експлуатацію;

– попереджувальне супроводження – діяльність, орієнтована на забезпечення адаптації системи до нових технічних можливостей.

Одна з проблем, що впливає на процес внесення змін, – це ступінь підготовки персоналу, здатного вносити необхідні зміни при виникненні  нерегулярних умов.

У зв'язку з тим, що майже кожні 8–10 років відбувається зміна архітектури комп'ютерів, мов програмування і операційних середовищ, виникають проблеми супроводу готових ПС і їхніх компонентів в новому середовищі або архітектурі, вирішення яких призводить до зміни або оновлення окремих елементів системи або системи повністю.

В цілому процес зміни (еволюції) ПС проводиться шляхом:

– аналізу початкового коду для внесення в нього змін;

– настроювання компонентів і системи на нові платформи;

– кодування і декодування даних при переході з однієї платформи на іншу;

– зміни функцій системи або додавання нових;

– розширення можливостей (сервісу, мобільності та ін.) компонентів;

– перетворення структури системи або окремих її компонентів.

Мета внесення змін в один компонент або в їхню сукупність – додавання старій ПС нового призначення в нових умовах застосування. Методи зміни ПС є засобом продовження життя успадкованих і застарілих програм. З теоретичної точки зору ці методи вивчені недостатньо, а на практиці багато програмістів розв’язують  задачі внесення змін в ПС постійно.

Наприклад, широкого кола фахівців торкнулася проблема зміни формату дати в 2000 році. Для систематичного перероблення функціонуючих програм з новими можливостями ОС, мов і платформ сучасних комп'ютерів тощо використовується сучасний аутосорсинг програмування (Індія, Росія, Україна та ін.).

Внесення змін до ПС можна розглядати як еволюційний шлях його розвитку. Еволюція ПЗ здійснюється такими зовнішніми методами обробки компонентів в розподіленому середовищі і внутрішніми методами, як зміна компонентів (Сом), інтерфейсів (Int) і/або систем. До внутрішніх методів еволюції належать методи реінженерії, рефакторинга і реверсної інженерії (рис.º1).

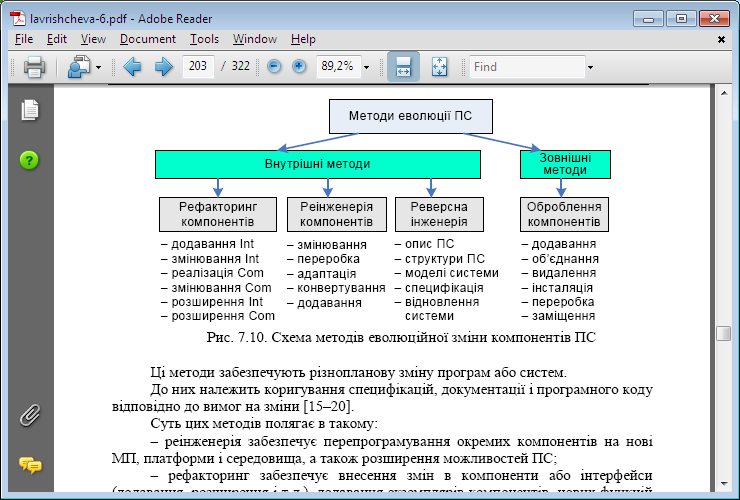


Рис. 1. Схема методів еволюційної зміни компонентів ПС

Ці методи забезпечують різнопланову зміну програм або систем.

До них належить коригування специфікацій, документації і програмного коду відповідно до вимог на зміни.

Суть цих методів полягає в такому:

– реінженерія забезпечує перепрограмування окремих компонентів на нові мови програмування, платформи і середовища, а також розширення можливостей ПС;

– рефакторинг забезпечує внесення змін в компоненти або інтерфейси (додавання, розширення тощо), додавання екземплярів компонентів, нових функцій або системних сервісів;

– реверсна інженерія означає повну переробку компонентів, а іноді і перепрограмування всієї системи.

**Реінженерія програмних систем**

**Реінженерія** (reengineering) – це еволюція програми (системи) шляхом її зміни з метою підвищення зручності її експлуатації, супроводу або зміни її функцій. Вона містить у собі процеси реорганізації і реструктуризації системи, переведення окремих компонентів системи в іншу, сучаснішу мову програмування, а також процеси модифікації або модернізації структури і системи даних. При цьому архітектура системи може залишатися незмінною.

Метод реінженерії**–**цільовий засіб отримання нового компонента шляхом виконання послідовності операцій внесення змін, модернізації або модифікації, а також перепрограмування окремих компонентів ПС. Реалізується сукупністю моделей, методів і процесів, що змінюють структуру і можливості компонентів з метою отримання компонента з новими можливостями. Нові компоненти ідентифікуються іменами, які використовуються при створенні компонентних конфігурацій і каркасів системи.

З технічної точки зору реінженерія – це вирішення проблеми еволюції системи шляхом зміни її компонентів, архітектури в середовищі, в якому компоненти розміщуються на різних комп'ютерах. Причиною еволюції може бути зміна мови програмування системи, наприклад, Fortran, Сobol та ін. з переходом на сучасні об'єктно-орієнтовані мови, такі, як Java або C++.

Проте з комерційної точки зору реінженерію часто вважають єдиним способом збереження успадкованих систем в експлуатації. Повна еволюція системи є дорогою або ризикованою процедурою продовження часу існування системи.

Порівнянну з радикальнішими підходами до вдосконалення систем реінженерія має такі переваги:

1. Зниження ризику при повторній розробці ПС. В той же час існує ризик отримання незадовільного результату при видаленні помилок в специфікації або при зміні функціональності деяких програм. Понизити виникаючі ризики можна за рахунок видалення помилок і покращення якості роботи змінених програм.

2. Зниження витрат за рахунок використання компонентів повторного використання при розробці нової ПС. Згідно з даними різних комерційних структур повторне використання в чотири рази дешевше, ніж нове розроблення системи.

Реінженерія застосовується для зміни ділових процесів, зниження кількості зайвих видів діяльності в них і підвищення ефективності окремих ділових процесів за рахунок впровадження нових програм або модифікації існуючих. Якщо бізнес-процес залежить від успадкованої системи, то зміни в ній повинні плануватися.

*Головна відмінність між реінженерією і новою розробкою* системи полягає в тому, що опис системної специфікації починається не з «нуля», а з розгляду можливостей старої успадкованої системи.

До основних процесів процесу реінженерії належать:

– переклад початкового коду в старій мові програмування на сучасну версію цієї мови або в іншій мові програмування;

– аналіз програм згідно з документованою структурою і функціональними можливостями системи;

– модифікація структури програм для нарощування нових властивостей і можливостей;

– розбиття системи на модулі для їхнього групування і усунення надмірності;

– зміна даних, з якими працює програма.

Причинами, що вимагають перетворення початкового коду програм в іншу мову, можуть бути:

– оновлення платформи апаратних засобів, на якій може не виконуватися компілятор мови програмування;

– недолік кваліфікованого персоналу для програм, написаних в мовах програмування, що вже не застосовують;

– зміна структури програми у зв'язку з переходом на нову стандартну мову програмування.

До операцій реінженерії належать:

– іменування змінних компонентів і їхня ідентифікація;

– розширення функцій існуючої реалізації компонентів;

– переклад мови компонента в нову сучасну мову програмування;

– реструктуризація  компонента;

– модифікація опису компонента і його даних.

# Рефакторінг компонентів

**Рефакторінг** розвивається в об'єктно-орієнтованому програмуванні у зв'язку з широким застосуванням інтерфейсів, шаблонів проектування і методів покращання коду. Розроблено бібліотеки типових трансформацій пошукових об'єктів (класів), які покращують ті або інші характеристики ПС.

Метод рефакторингакомпонента – це цільовий спосіб отримання нового компонента на базі існуючого, який містить у собі операції модифікації (зміна, заміщення, розширення) компонентів і інтерфейсів. Мета методу – перетворення складу компонентів ПС або зміна окремого компонента системи для додання йому нових функціональних і структурних характеристик, що задовольняють вимоги конфігурації. Метод містить у собі сукупність моделей, методів і процесів, що застосовуються до певних класів об'єктів і компонентів для отримання нових або змінених об'єктів-компонентів з метою підвищення якісних характеристик ПС або додавання нових можливостей.

Процес рефакторингу орієнтується на отримання нових компонентів, які містять у собі такі операції з організації проведення змін:

– додавання нової реалізації для існуючого і нового інтерфейсу;

– заміна існуючої реалізації новою з еквівалентною функціональністю;

– додавання нового інтерфейсу (за наявності відповідної реалізації);

– розширення існуючого інтерфейсу для нових системних сервісів у компонентному середовищі.

Кожна операція рефакторингу – базова, атомарна функція перетворення, що зберігає цілісність компонента, тобто правила, обмеження і залежності між складовими елементами компонента, що дозволяють розглядати компонент як єдину і цілісну структуру з своїми властивостями і характеристиками.

Після виконання операцій рефакторингу компоненти повинні бути ідентичні функціям початкового компонента. У разі докорінної зміни групи компонентів системи шляхом внесення нових функцій система набуває нової функціональності.

Операції над компонентами задовольняють умови:

– об'єкт, одержаний внаслідок рефакторингу, – це компонент з відповідними властивостями, характеристиками і типовою структурою;

– операція не змінює функціональність компонента, і новий компонент може застосовуватися в раніше побудованих компонентних системах;

– перебудова компонентів, а іноді і перепрограмування проводиться в процесі реверсної інженерії

# Реверсна інженерія

Методи реверсної інженерії, які розроблено в середовищі об'єктно-орієнтованого програмування, ґрунтуються на виконанні базових операцій візуалізації (visual) і вимірювання метрик (metric) ПС в рамках моделі, яка пропонує такі цілі:

– забезпечення високої якості системи і перезасвідчення її розміру, складності і структури;

– пошук ієрархії класів і атрибутів програмних об'єктів з метою успадкування їх в ядрі системи;

– ідентифікація класів об'єктів з визначенням розміру і/або складності всіх класів системи;

– пошук шаблонів (патернів), їхня ідентифікація, а також фіксація їхнього місця і ролі в структурі системи.

Цей підхід орієнтується на індустріальні системи в мільйон рядків коду з використанням метричних оцінок характеристик системи. Він вирішує генерацію тестів для перевірки кодів, а також проведення метричного аналізу системи для отримання фактичних значень внутрішніх і зовнішніх характеристик системи.

Після аналізу системи будується модель, яка містить у собі список класів і патернів системи, які можуть модифікуватися і перепроектуватися, і тим самим забезпечувати процес еволюції системи. Якщо деякий клас погано спроектовано (наприклад, багато методів, пусті коди) або система не виконує необхідну роботу, то проводиться збирання інформації для зміни цієї моделі.

У даному підході дії з візуалізації системи висвітлюють на екрані у вигляді ієрархічного дерева, вузли якого відображають об'єкти і їхні властивості, а відношення задаються контурами команд фрагментів програм. При цьому застосовується таблиця метрик, в якій знаходяться відомості про метрики класів об'єктів (число класів, атрибутів, підкласів і рядків коду), метрик методів об'єктів (кількість параметрів виклику, повідомлень і т.п.), метрик атрибутів об'єктів (час доступу, кількість доступів в класі і т.п.).

У процесі візуалізації  збираються метричні данні про систему. Якщо реально визначено всі дані в різних фактичних метриках ПС, то виконується оцінка якості і розроблюється план перебудови застарілої системи на нову систему з отриманням тих же можливостей або ще і додаткових.

# Стандарт ISO/IEC 11404–96 з незалежних від мов типів даних

Мета даного стандарту та гармонізованого ГОСТ 30664-99 полягає в тому, щоб забезпечити не тільки опис типів даних в стандартній мові LI (Language Independent) і їхню генерацію, а й перетворення типів даних мов програмування у LI-мову, і навпаки. Стандарт пропонує спеціальні правила і характеристичні операції генерації примітивних типів даних і об'єднань LI-мови в простіші структури даних мов програмування, а також визначення параметрів інтерфейсу засобами мов IDL, RPC і API.

Незалежні від мов програмування типи даних стандарту розділено на примітивні, агрегатні і такі, що згенеровані (рис. 2.).

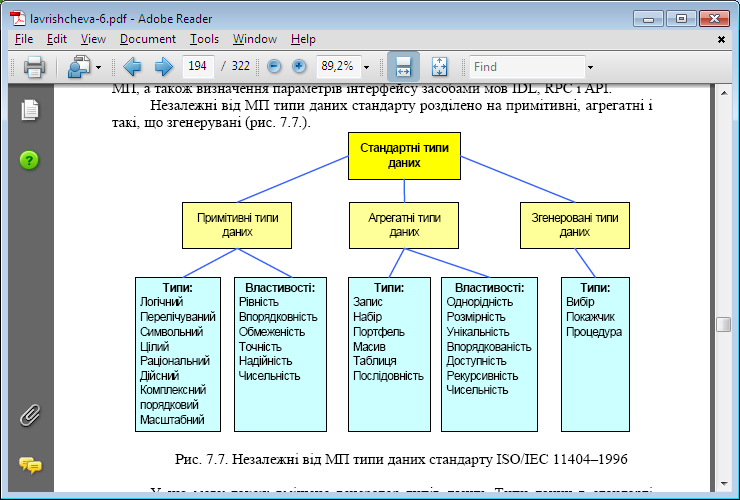


Рис. 2. Незалежні від мови програмування типи даних стандарту ISO/IEC 11404–1996

У цю мову також вміщено генератор типів даних. Типи даних в стандарті описуються в LI-мові, яка є більш загальною мовою, на відміну від конкретних засобів опису типів даних мов програмування.

Стандарт містить у собі всі існуючі типи мови програмування і загальні типи даних, орієнтовані на генерацію інших типів даних. Стандарт складається з розділів (рис. 3): об’ява (declaration) типів даних, об’явлені типи даних; об’явлені  генератори.

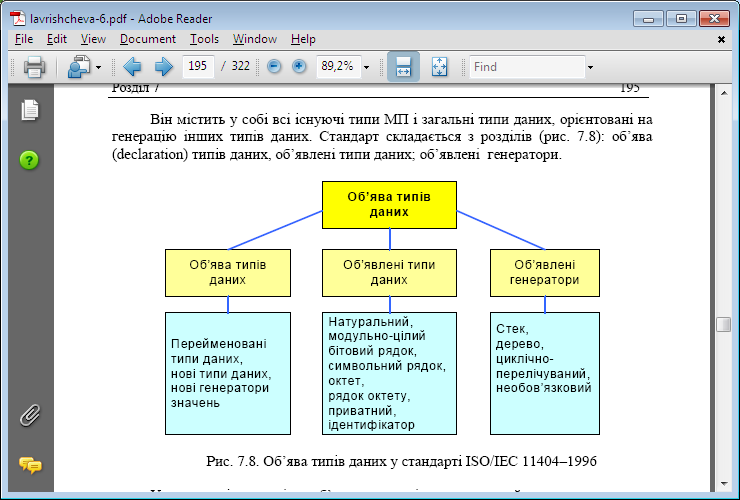


Рис. 3. Об’ява типів даних у стандарті ISO/IEC 11404–1996

У стандарті є розділ  об’явлення типів даних, перейменування тих, що існують;  об’явлення нових генераторів, значень і результатів. Кожен тип даних має шаблон, що містить у собі опис і специфікатор типу даних, значення в просторі значень, синтаксичний опис і операції над типами даних.

Для об’явленого типу даних задається шаблон, що вміщує синтаксичний опис, специфікатор типу даних, значення в просторі значень і операції над цим типом даних.

Засобами мови LI описуються параметри виклику як елементи інтерфейсу, необхідні при зверненні до стандартних сервісів і готових програмних компонентів.

Запропоновані в стандарті рекомендації, а також засоби опису типів даних і методів їхнього перетворення є універсальними.

# Перетворення форматів даних

Програми, розташовані на різних типах комп'ютерів, передають один одному дані через протоколи, їхні формати перетворюються до формату даних приймаючої серверної платформи (так званий *маршаллинг даних*) з урахуванням порядку і стратегії вирівнювання, прийнятої на цій платформі. *Демаршаллинг даних* – це зворотне  перетворення даних (тобто отриманого результату) до виду клієнтської передавальної програми. Якщо серед переданих параметрів оператора виклику містяться нерелевантні типи або структури даних, які не відповідають параметрам викликаного об'єкта, то проводиться пряме і зворотне їхнє перетворення засобами  стандарту або мов програмування.

До засобів перетворенняданих і їхніх форматів належать:

 – стандарти кодування даних (XDR – eXternal Data Representation, CDR –Common Representation Data), NDR – Net Data Representation) і методи їхнього перетворення;

– мови програмування і механізми звернення компонентів один до одного;

– мови опису інтерфейсів компонентів – RPC, IDL і RMI для передачі даних між різними компонентами.

На кожній платформі комп'ютера використовуються угоди про кодування символів (наприклад, ASCII), про формати цілих чисел і чисел з плаваючою точкою (наприклад, IEEE, VAX і ін.). Для представлення цілих типів, як правило, використовується додатковий код, а для типів float і double – стандарт ANSI/IEEE та ін.

Порядок розташування байтів залежить від структури платформи (Big Endian або Little Endian)  від старшого до молодшого байту і від молодшого  до старшого байту. При передачі даних з однієї платформи на іншу враховується можливий незбіг порядку байтів. Маршаллинг даних підтримується такими стандартами.

**XDR–стандарт** містить у собі  мову опису структур даних довільної складності і засоби перетворення даних, що передаються на платформи (Sun, VAX, IBM і ін.). Програми в мові програмування можуть використовувати дані в XDR-форматі, не зважаючи на те, що компілятори вирівнюють їх в пам'яті машини по-різному.

У XDR-стандарті цілі числа з порядком «від молодшого» зводяться до порядку байтів «від старшого» і назад. Перетворення даних – це кодування (code) або декодування (decode) XDR-процедурами форматування простих і складних типів даних. Кодування – це перетворення з локального уявлення в XDR-уявлення і запис в XDR-блок. Декодування – це читання даних з XDR-блоку і перетворення в локальне уявлення заданої платформи.

Вирівнювання даних – це розміщення значень базових типів з адреси, кратної дійсному розміру в байтах (2, 4, 8, 16). Межі даних вирівнюються за  найбільшою довжиною (наприклад, 16). Системні процедури оптимізують розташування полів пам'яті під складні структури даних і перетворюють їх до формату приймальної платформи. Оброблені дані декодуються назад до виду формату передавальної платформи.

**CDR-cтандарт**середовища CORBA забезпечує перетворення даних у формати платформи, що їх передає або приймає. Маршаллинг даних виконує інтерпретатор TypeCode і брокер ORB. Процедури перетворення складних типів вміщують:

– додаткові коди для представлення цілих чисел і чисел з плаваючою точкою (стандарт ANSI/IEEE);

– схему вирівнювання значень базових типів в середовищі компілятора;

– базові типи (signed і unsigned) в IDL, а також плаваючому типі подвійної точності та ін.

Перетворення даних виконуються процедурами encoder ( ) і decoder ( ) інтерпретатора TypeCode, який використовує базові примітиви при вирівнюванні інформації і розміщенні її в буфері. Для складного типу визначається розмір і межі вирівнювання, а також їхнє розміщення в таблиці з індексами значень TCKind, які використовуються при ініціалізації брокера ORB.

**ХМL-стандарт**забезпечує усунення неоднорідності у взаємозв'язках компонентів у різних мовах програмування за допомогою XML-формату даних, який враховує різні платформ і середовища. Проміжні середовища (CORBA, DCOM, JAVA та ін.) мають у своєму складі спеціальні функції, аналогічні XML – альтернатива сервісам CORBA в плані забезпечення взаємозв'язків різномовних програм.

XML має різну системну підтримку: браузер Internet Explorer для візуалізації XML-документів, об'єктна модель DOM (Document Object Model) для відображення XML-документів і інтерфейс IDL в системі CORBA.

Тексти в XML-стандарті описуються у форматі ASCII, що дає можливість ефективніше застосовувати їх при обміні даними. XML використовується для кодування типів даних за допомогою файлових форматів. При необхідності переходу програмної системи до XML-стандарту проводиться переформатування даних  у формат XML,  і навпаки.

Таким чином,  XML-мова дозволяє представляти об'єкти для різних  моделей на єдиній концептуальній, синтаксичній і семантичній основі. Він не залежить від платформи і середовища моделі взаємодії компонентів прикладного рівня. XML спрощує обробку документів, роботу з БД за допомогою стандартних методів і засобів.

# Перетворення даних з баз даних

Перетворення даних БД пов'язане з різницею логічних структур даних, а також з такими проблемами:

1) багатомодельність представлення даних (ієрархічні, мережні, реляційні) в різних БД і СКБД. Проблема розв'язується шляхом переходу до реляційної моделі даних і СКБД, яка є потужним математичним апаратом, який ґрунтується на теорії множин і математичній логіці. Ця модель складається із структурної, маніпуляційної і цілісної частин. У цих частинах, відповідно, фіксується структура даних, опис програм в SQL-мові і вимоги до цілісності. Ієрархічні або мережні моделі даних загалом не підтримують цілісність, тому при переході від них до реляційних БД виникає порушення цілісності даних;

2) різниця в логічних структурах даних, в довідниках, класифікаторах і в системах кодування інформації. Проблема викликана тим, що логічна структура даних або концептуальна схема БД припускають проектування нової структури БД при зміні наочної області або при переході до  нового типу СКБД. При цьому зіставляються дані старої і нової БД і змінюється довідкова інформація і класифікатори;

3) використання різних мов для представлення текстової інформації. Проблема визначається різномовними текстовими представленнями інформації в БД. У старих БД використовувалася, як правило, одна мова, а в нових їх може бути декілька, тому для зберігання даних з простим доступом до текстових даних встановлюється відповідність текстових даних, записаних в різних мовах;

4) різні типи СКБД і постійний розвиток даних БД в процесі експлуатації. Проблему можна сформулювати як метод зберігання і обробки різних даних, викликаних специфікою СКБД ієрархічного, мережного і реляційного типів. Наявність явної несумісності типів і структур цих моделей даних, різні мови маніпулювання даними призводять до того, що не можна згенерувати мовою старої СКБД скрипти для перенесення даних і подальшого запуску БД в середовище іншої СКБД. Кожна СКБД забезпечує внесення змін в БД, які певною мірою змінюють і концептуальну модель даних, якщо до неї вносять нові об'єкти. Внесені зміни повинні відображатися в довідниках і класифікаторах, що забезпечує перенесення даних із старої БД до  нової з урахуванням поточних змін.

**Перетворення даних.**Шляхи вирішення наведених проблем.

При тривалій промисловій експлуатація систем, що працюють з БД, можуть змінюватися прикладні програми і дані, якщо в систему введена нова БД, а частина раніше визначених даних перенесена до нової БД.

Це спричиняє необхідність доопрацювання прикладних програм доступу, щоб пристосувати їх до зміненої структури нової БД або до старої БД. Для перенесення даних із старої БД до  нової створюються скрипти або DBF-файли, які розміщуються в транзитній БД для перенесення до нової БД. Якщо виявиться, що процес зведення структури транзитної БД до нової виявився недоцільним, то розроблення нової БД проводиться «з нуля». При цьому довідники і класифікатори доповнюються новими даними, що з'явилися.

Проблеми перетворення даних при використанні різних СКБД виникають також через те, що дані мають різні способи зберігання, серед яких можуть опинитися несумісні типи даних або доступ до даних здійснюється різними мовами маніпулювання.

Перетворення даних може проводитися кілька разів шляхом створення спеціальних скриптів і файлів з урахуванням раніше введених даних, без їхнього дублювання і коректного зведення несумісних типів даних. Можуть виникнути помилки, пов'язані із зміною форматів даних, доповненням старих довідників новими даними і т.п.

Процеси перетворення даних базуються на використанні:

* методу 1, що переносить дані із старої БД до  транзитних файлів, а потім заносить ці файли до транзитної БД;
* методу 2для обробки даних в транзитній базі при зміні кодування даних, приведенні відповідності між структурами старої і нової БД, а також кодів довідників і класифікаторів;
* методу 3 для системного перенесення даних з транзитної бази до основної БД з перевіркою перетворених даних.

Перший метод – найбільш безболісний для користувачів і розробників. Другий метод є створенням нового проекту системи із заданою моделлю даних. При третьому методі – система створюється заново, до нової БД можуть заноситися успадковані дані із старої БД. Оскільки структури БД можуть виявитися різними, то, як правило, створюються тимчасові застосування, в яких здійснюються необхідні перетворення даних при перенесенні до нової БД.

При застосуванні першого і другого методів структура старої БД зберігається і ніякого перетворення даних, відповідності довідників і класифікаторів не вимагається, оскільки вони використовують єдиний формат зберігання даних.

**Файли передачі даних між різними БД.**Проблема перетворення і перенесення даних між різними СКБД розв'язується через:

1) спеціальний драйвер (дві СКБД з'єднуються один з одним і безпосередньо передають дані, використовуючи інтерфейс);

2) транзитні файли, в які копіюються дані із старої БД для перенесення  їх у нову БД.

Перетворення і перенесення даних з різних БД до нових БД наведено на рис. 4.

У разі використання драйвера дві СКБД взаємодіють безпосередньо і передають дані, використовуючи певний інтерфейс і спеціальні програми взаємодії двох СКБД, при яких друга СКБД розуміє результати виконання запитів на мові маніпулювання даними першої СКБД,  і навпаки. Дані на виході першої СКБД є даними на вході другої СКБД в мові маніпулювання даними другої СКБД, такі дані можуть бути внесені до транзитної БД.

Такий метод складний в реалізації і вимагає постачання програм перенесення даних з інших СКБД, які прив'язано до старої і нової СКБД. Тому другий метод перенесення даних між різними СКБД більш прийнятний.

У другому випадку дані із старої БД переносяться до  транзитних файлів, SGL-скриптів, DBF-файлів з наперед заданими форматами даних, які пересилаються до нової транзитної БД через мережу за допомогою спеціальних утиліт або засобів нової СКБД.

Якщо друга СКБД реляційного типу, то дані в транзитних файлах перетворяться до табличного вигляду. Якщо перша СКБД не реляційна, то дані повинні бути приведені до табличного вигляду і першої нормальної форми.

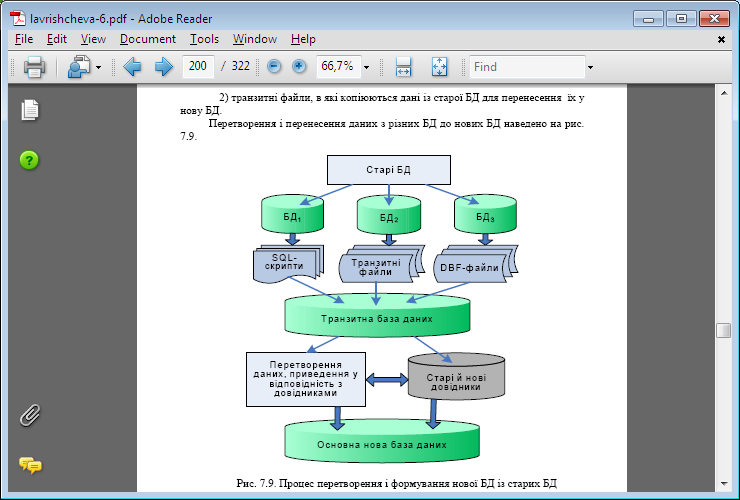


Рис. 4. Процес перетворення і формування нової БД із старих БД

Подальша нормалізація даних і зведення їх до структури нової БД здійснюється в транзитній БД з використанням 3- або 4- нормальної форми для подання структур даних. Кожна вища форма нормалізації містить у собі як підмножину нижчу форму, наприклад, першу нормальну форму у вигляді скалярних значень.

Іншими словами, відношення знаходяться в першій нормальній формі, якщо вони зберігаються в табличному вигляді (всі елементи в рядку таблиці розташовані в строго певній послідовності) і кожний елемент таблиці містить у собі  тільки атомарні значення (елемент не є множиною).

Відношення знаходиться в третій нормальній формітоді і тільки тоді, коли кожний кортеж складається із значення первинного ключа, що ідентифікує деяку суть, і набору пустих значень або значень незалежних атрибутів цієї суті. Тобто відношення знаходиться в третій нормальній формі, коли неключові атрибути – взаємно незалежні, але залежать від первинного ключа.

Два або декілька атрибутів – взаємно незалежні, якщо жодний з них не залежить функціонально від комбінації решти атрибутів. Подібна незалежність припускає, що кожен атрибут можна оновлювати незалежно від інших.

Процес нормалізації відношень дозволяє позбавитися проблем, які можуть виникнути при оновленні, внесенні або видаленні даних, а також при забезпеченні цілісності даних. Структури старих БД не завжди можна привести до третьої нормальної форми, тому потрібно, щоб дані, що знаходяться в транзитних файлах, існували хоч би в першій нормальній формі і належали до реляційної моделі.

Як  уніфікований формат транзитних файлів використовують формат DBF-файлів, оскільки багато СКБД, такі, як DB2,FохРго і деякі інші, зберігають дані в таких файлах, тим самим немає потреби у початковому перенесенні даних із старої СКБД до транзитних файлів. Більшість СКБД, формат зберігання даних яких відрізняється від формату DBF-файлів, забезпечується утилітами або драйверами, що дозволяють перенести дані в такий формат.

***Контрольні запитання для самоперевірки***.

1. Визначте цілі і завдання зміни ПС при супроводі
2. Охарактеризуйте проблеми, що виникають при супроводі системи.
3. Визначте основні завдання реінженерії ПС.
4. Чим відрізняється рефакторинг компонентів від реінженерії?
5. Визначте основні операції реверсної інженерії ПС.
6. Визначте проблеми перетворення форматів даних.
7. Які методи перетворення даних БД існують?

**Література**

1. Бабенко Л.П., Лавріщева К.М*.* Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001.– 269с.
2. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.
3. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
4. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.