Валерій Сидоренко,
доцент кафедри інформаційних технологій УАДУ,
кандидат економічних наук

Загальна концепція побудови інформаційних систем забезпечення державного управління

Сучасний етап державно-правового будівництва в Україні характеризується інтенсивним пошуком принципово нових рішень на шляху глибоких перетворень у всіх сферах життя держави і суспільства.

У зверненні Президента України Леоніда Кучми до Верховної Ради України 4 березня 1995 р. наголошувалося на необхідності опрацювання нової регіональної політики, суть якої полягає в розширенні самостійності та підвищенні відповідальності місцевих органів влади і управління в життєзабезпеченні та розвитку територій, міст, селищ та інших населених пунктів з урахуванням місцевої специфіки, соціально-економічних особливостей і культурно-історичних традицій. Така децентралізація влади посилить загальнодержавні інститути влади, оскільки дасть змогу зосередитися на ключових проблемах державного будівництва.

Згадана політика потребує системи цілеспрямованих і взаємопов'язаних законодавчих і нормотворчих актів. її основним практичним інструментом має стати чіткий законодавчий розподіл повноважень, відповідальності, а отже, і фінансово-економічної бази між загальнодержавним, територіальним і місцевим рівнями управління.

Нова регіональна політика - це відмова від взаємодії між рівнями адміністративної підпорядкованості, за якої одне й те саме питання вирішується на всіх рівнях шляхом бюрократизованих процедур погодження згори донизу і заміна їх на чітко визначений розподіл компетенції кожного рівня. Принципово важливим у децентралізації влади є положення про невтручання держави в особі її центральних органів у сферу суверенних прав міст, областей чи районів. Втручання допускається лише у виняткових випадках, зазначених у Конституції.

Такий підхід передбачає перерозподіл компетенції між державою і регіонами. За державою зберігається жорсткий контроль за регіонами в питаннях додержання ними законів та їх фінансовою діяльністю, а на регіони внаслідок розширення самостійності лягає тягар делегованих державою повноважень, який має компенсуватися наданням місцевим органам еквівалентного обсягу фінансових ресурсів.

Отже, система управління в Україні будується на трьох рівнях (шарах): загальнодержавному, територіальному і місцевому, кожен з яких одночасно є і автономним, і залежним.

Шар місцевого управління (нижній за ієрархією) автономний у межах делегованих йому повноважень і залежить від середнього шару управління, на який покладено контроль за не перевищенням і чітким виконанням підпорядкованим рівнем управління наданих йому повноважень.

Аналогічно взаємодіють середній та вищий шари управління в державі.

Шар управління на державному рівні контролюється законодавчою гілкою влади в межах визначених Конституцією України повноважень.

Незалежно від рівня (шару) управління процес прийняття управлінського рішення передбачає такі етапи (можливо, розтягнуті в часі):

- збирання (накопичення) інформації, необхідної для прийняття рішення;
- розробка проекту рішення (обробка інформації, аналітичні розрахунки, аналіз можливих наслідків прийняття рішення, коректність рішення щодо законодавчої бази, відповідність рівню повноважень того, хто приймає рішення);
- прийняття рішення (формування документа, в якому відображено суть прийнятого рішення).

Згідно зі згаданою технологією прийняття управлінських рішень розглянемо запропоновану автором концепцію побудови інформаційних систем забезпечення державного управління.

Головна ідея цієї концепції базується на постулаті "велика інформаційна система дуже складна і тому, щоб вона була керованою, її потрібно звести до малої". Одна особа не в змозі охопити все - наприклад, створювати і обслуговувати дуже велику систему. Можливо, населення інших планет здатне тримати в пам'яті кожну деталь систем, які мають тисячі частинок і мільйони взаємозв'язків, однак людині, яка живе на Землі, це, напевно, не під силу. Тобто пропонується втілити в життя принцип "розподіляй і володарюй", який вже реалізований у мережах комутованих пакетів, розподілених системах, самокерованих колективах. Ця сама ідея закладена і в тришарову архітектуру додатків (applications) в системах клієнт/сервер. . Отже, створюючи нову систему, необхідно знайти шлях поділу великих додатків на серію малих. Загальна система може залишатися великою, але її потрібно звести до багатьох малих частин, кожна з яких зрозуміла. Тому пропонуються два механізми розподілу і управління складністю у великих додатках: абстрагування та інкапсуляція.

Абстрагування - певний аналітичний метод для поділу систем на багато шарів, деталей. Абстрагування перебуває в центрі ефективної архітектури додатків.

Інкапсуляція - процес об'єднання інформації та поведінки в деяку нову сутність, яка дістала назву компонент.

Абстракція - це опис, який ідентифікує істотні риси об'єкта, водночас приховуючи його конкретні деталі. Головним принципом створення якісного проекту системи є розробка її архітектури, яка забезпечить велику кількість рівнів абстрагування. Перехід до узагальнення означає пересування вверх на один рівень абстракції.

Архітектура розшарування, яка базується на абстракції, має три основні характеристики.

Чітко визначені шари. Якщо ця архітектура точно базується на абстракції, то можлива така побудова шарів один над одним, щоб кожен новий шар забезпечував більш складну і комплексну абстракцію порівняно з шаром, розміщеним нижче.

Формальні і явні інтерфейси *між* ***конкретними шарами.*** Кожен шар має змогу звертатися за послугами до шару, розміщеного нижче. Інтерфейси визначають високорівневі дії, що активізують дії, які виконуються на розміщеному нижче шарі. Це аналогічно тому, як мозок видає команди - підняти праву ногу, опустити її попереду лівої. Такі високорівневі операції перетворюються на специфічні нервові сигнали, які активізують групи м'язів і забезпечують зворотний зв'язок з органами чуття вашого тіла. Проте всі ці деталі приховані високорівневим інтерфейсом між мозком і тілом. Отже, інтерфейс шару великою мірою визначає цей шар. Згадані операції доступні для розміщеного вище шару, а інтерфейс точно визначає, які аспекти функцій всередині

цього шару є видимими і доступними для зовнішнього середовища. Наприклад, раціональна свідомість не має інтерфейсних функцій, які допоможуть їй контролювати биття серця. У результаті контроль пульсу розглядається як функція, непідлегла свідомості, оскільки вона відсутня в інтерфейсі.

Приховані і захищені деталі всередині кожного шару. Приховування деталей є важливою функцією як абстрагування, так і інкапсуляції. Наприклад, ваш раціональний мозок звичайно "думає" в термінах високорівневих дій, які може виконати тіло: підняти руку, зробити крок тощо. Водночас ваше тіло буквально забороняє раціональній свідомості контролювати його на більш детальному рівні. Проілюструємо це. Коли ви починаєте бігти, ділянка вашого мозку, тіло і нервова система змушують серце збільшити ритм. Після зупинки серце починає знову битися повільніше. Це автоматична поведінка, яку ви не можете контролювати раціональною свідомістю, навіть якщо забажаєте.

Три концентричні кола, зображені на рисунку, являють собою конкретні шари зростання абстракцій в пропонованій архітектурі додатків.

Найбільш деталізованим є **шар бази даних** (БД), розмішений у центрі.

Наступний, вищий рівень абстракції - **шар правил управління**, який розмішується зовні шару управління БД.

І останній, найвищий рівень абстракції - **шар документа** - перебуває на зовнішньому боці кола. На цьому рисунку зовнішнє коло є шаром, який бачать користувачі, внутрішнє - тим, що глибоко приховане в системі.

Тришарова архітектура додатків передбачає необхідність захищати кожен шар від конкретних деталей, які є змістом шару, що лежить нижче. Наприклад, шар правил управління має бути захищеним концептуально і механічно від шару управління БД.

Взагалі існують дві основні переваги використання абстрагування за рахунок захисту кожного шару від деталей інших шарів.

Спрощена розробка додатків. Концептуально робота розробника в одному шарі спрощена тому, що він не бажає знати чи думати про конкретні деталі шару, що лежить нижче. (Раціональна свідомість не турбується про те, які нервові волокна збуджуються.)

Підвищена безпека і секретність. Розробник конкретного шару фізично не в змозі контролювати шар, що лежить нижче, за деталями цього рівня. (Раціональна свідомість не може збуджувати окремі нейрони, навіть якщо цього захоче). Тобто побудова абстракцій ваших додатків має підвищити безпеку і секретність таких систем.

Рівні абстракції в тришаровій архітектурі додатків

Розглянемо кожен з шарів окремо з точки зору їх відповідності різним управлінським рівням у ієрархічній системі прийняття рішення.

Шар управління БД відтворює рівень клерків в установі старого типу. Клерки вели індивідуальні книги обліку, підшивали різні папери в папки і працювали з індексними картками. Вони керувалися посадовими інструкціями (наприклад, переносили записи з одного документа до іншого тощо). Клерк, що ретельно заповнював документ деякою інформацією (транзакції БД), знав дуже мало про дії керівництва, яке визначало його роботу. Він зосереджувався на перенесенні конкретних даних у відповідне місце. Для клерків система реєстрації та обліку була цілим світом.

У тришаровій архітектурі додатків центром додатку є шар управління БД, в якому все побудовано на основі інформації. Значною мірою цей шар може бути однією конкретною частинкою такої системи, яка може існувати самостійно. Інформація має сенс і значення незалежно від форми подання правил управління або додатків.

Правила управління відповідають функціям керівництва середньої ланки. Виконуючи правила управління, керівники середньої ланки постійно генерують потік деталізованих транзакцій клеркам своєї організації. Вони працюють на більш високому рівні абстракції, і тому не усвідомлюють конкретні деталі завдань, які вирішують клерки. Так само сприймають правила управління в бюрократичних системах і керівники середньої ланки, і клерки незважаючи на те, що вони працюють на різних рівнях деталізації.

Аналогічно процеси правил управління базуються на шарі управління БД в тришаровій архітектурі додатків. У міру того, як інші шари додатка генерують вимоги до процесів управління, шар правил управління виконує такі дії:

- вирішує потрібні завдання;
- приймає рішення;
- запускає інші завдання в шарі правил управління та в решті шарів.

Шар документа можна порівняти з діями керівництва вищої ланки, яке видає головні розпорядження і визначає зміни в правилах, використовуючи заздалегідь визначені форми (додатки) і запити про виконання конкретних процесів за правилами управління. Кожен запит від старшого керівника активізує роботу керівників середньої ланки (процеси Правил управління) і клерків (транзакції БД). Однак керівництво вищої ланки не вникає в деталі цих наступних дій.

А зараз розглянемо, які функції має підтримувати і за що відповідати кожен з шарів у тришаровій архітектурі додатків (табл. 1).

Таблиця 1

Функції шарів в архітектурі додатків

Шар	Відповідальність	Функції	Інструментальні засоби
Документ	Зрозумілий, ефективний інтерфейс	Подання, навігація, маніпулювання та	Графічні засоби і мови
Правила управління	Політика: правила та евристичні процедури прийняття	Прийняття рішень, проведення політики,	С, СОБОЛ, процесори правил, Basic
БД	Узгоджені, захищені дані	Забезпечення узгодженості, секретності,	БД, мови БД

Верхній шар - це шар документа, який часто вважають шаром "додатків робочого

столу". Шар документа відповідає за забезпечення інтерфейсу користувача для всієї системи; Назва цього шару викликає найбільше нарікань. Користувачі вважають програми, які запускаються на їхніх робочих столах, додатками або інструментальними засобами. Однак розробники всієї системи вважають правила управління і компоненти баз даних частинками додатків. При остаточному аналізі, вибираючи назву цього шару, виходили з того, що дійсно видається користувачеві на екран - документів. Документ може бути формою, графіком, пояснювальною запискою або частиною електронної пошти. Цим і пояснюється назва.

Документи відповідають за зрозумілість і ефективність. Зрозумілість - це те, чим займається Graphical User Interface (GUI): подання інформації в зрозумілій формі; надання можливості користувачеві контролювати комп'ютер без вивчення складних команд. Ефективність - це здатність додатків допомагати користувачам виконувати роботу швидко.

Шар документа відповідає ще й за такі функції.

Навігація. Документ надає меню, форми і структуру команд, які дають змогу користувачам знайти те, що їм потрібно: команду для запуску чого-небудь або звіт, який потрібно надрукувати.

Подання. Документ виводить інформацію в різних формах, включаючи графіки, звуки, слова і числа.

Маніпулювання. Документ стола може створювати і змінювати інформацію відповідно до потреб користувача.

Аналіз. Комбінуючи функції подання і маніпулювання, документ дає змогу користувачеві виконувати аналіз "що буде, якщо" для прийняття рішення, відповіді або результату.

У загальному випадку шар документа перетворює персональний комп'ютер на електронний робочий стіл. Два інших шари вводять інформацію в цей робочий стіл, і він, отримуючи інформацію, стає тим, з чим доведеться працювати користувачеві. Саме те, що шар документа надає можливість користувачеві працювати з даними і змінювати їх, і робить системи клієнт/сервер такими зручними для користувачів.

Шар правил управління відповідає за політику організації. Політика - це дещо більше, ніж правила. Правило є точним виразом, звичайно, у формі "якщо, тоді...". На практиці багато рішень, які приймаються на рівні правил управління, не мають такої чіткої форми. Програми цього шару базуються на евристичних процедурах, - лінії поведінки, яку досить часто формулюють у ймовірнісних термінах. Наприклад, якщо підприємство сплачує за більшістю рахунків своєчасно, йому можна дозволити дещо збільшити кредит. Вислови "більшість рахунків" і "дещо збільшити" не дають змоги перетворити це твердження на точне правило. Разом з тим легко уявити процес (забезпечення) правил управління, які відтворюють подібну політику, використовуючи комбінування процентного аналізу, тенденцій і, від випадку до випадку, - запитів на втручання людини. Таким чином, шар правил управління відповідає за правила і евристичні процедури. Більш того, він реалізує правила і евристичні процедури у формі рішень у трьох значних за обсягом категоріях.

Формальні рішення припускають точні запити на перевірку повноважень. (Має ця людина права на певні пільги? Можливе виконання цих робіт до певного часу?) У таких випадках процес на рівні правил управління приймає конкретне рішення або відповідає на отриманий запит.

Рішення щодо проведення політики маються на увазі і є беззастережними. Запитання може бути і не заданим, але шар правил управління обов'язково приймає певні

безумовні рішення. (Керівник структурного підрозділу не має права приймати рішення, які перевищують його повноваження. Інформацію про підприємства, які не виконують зобов'язань, не можна видаляти з БД.) Подібна політика проводиться постійно - навіть тоді, коли ніхто спеціально не робить запитів за конкретними правилами.

Рішення з координації і управління ресурсами також припускаються і є беззастережними. (Закінчувати реєстрування на семінар, якщо не залишилось вільних місць. Давати позитивні відповіді тільки в разі реальної можливості їх виконання.) Управління ресурсами базується на рішеннях типу "якщо, у такому випадку". Отже, рішення з управління ресурсами впливають на розподіл безпосередньо ресурсів, а не дають звичайні відповіді "так/ні" на подані запити.

Шар управління БД відповідає за підтримку погодженості і захищеності інформації. Ґрунтовно розроблений шар управління БД підтримує захист інформації і погодженість даних, одночасно забезпечуючи високу продуктивність.

Захист інформації. Головне завдання шару управління БД - забезпечення секретності і збереження даних. Система ніколи не повинна випадково втрачати інформацію, яка в ній міститься. Тому потрібні ретельно продумані процедури з дублювання інформації, дорогі накопичувачі на магнітних стрічках і запам'ятовуючі пристрої надвеликої ємності. Крім того, цей шар має слідкувати за тим, щоб доступ до тієї чи іншої інформації отримували лише ті, в кого є на це повноваження.

Погодженість даних. Шар управління БД забезпечує осмисленість інформації, яка в ній зберігається. Для досягнення цієї мети шар управління БД зобов'язаний зробити інформацію доступною, коли в цьому виникає потреба, приймати нову інформацію тільки від людей, уповноважених вносити зміни, і форматувати нові дані у вигляді, погодженому з інформацією, яка вже є в БД. Найголовніше те, що проектування шару управління БД має забезпечувати погоджені відповіді як на поточні запитання, так і на запитання, що можуть виникнути в майбутньому.

Люди, які працюють з комп'ютерами, вільно використовують поняття "інтерфейс" у розмовах про системи. Є багато видів інтерфейсів: між людьми, графічні, між шарами в деякій архітектурі, між додатками тощо.

У тришаровій архітектурі додатків інтерфейс - це і конкретний простір між суміжними компонентами у додатку, і конкретний пристрій, за допомогою якого ці компоненти взаємодіють. Значною мірою інтерфейс розкриває конкретний зовнішній вигляд архітектурного компонента. У ролі службового протоколу для користувачів окремих компонентів інтерфейс виконує такі функції:

- повідомляє компонентам, що робити;
- постійно з'ясовує поточний стан конкретних компонентів;
- приймає результати операцій, про які був запит.

Фактично в деякому об'єктно-орієнтованому світі цей інтерфейс може стати всемогутнім. Загальним сенсом об'єктної орієнтації є інкапсуляція даних і конкретних програм, які працюють на цих даних, в межах окремого об'єкта. Тобто робота з конкретними даними можлива лише через інтерфейс до відповідних об'єктів. Не тільки повні шари, а й конкретні компоненти, що утворюють ці шари, формуються як об'єкти. У цьому розумінні інтерфейси є магічними ключами до конкретних шарів, а також до всіх конкретних компонентів усередині *їх.*

У табл. 2 подана модель для проектування кожного шару додатків. Як видно з таблиці, будь-який шар є незалежною сутністю з власним фокусом. Одним з ключових

моментів, що проходить крізь усю тришарову архітектуру, є важливість підтримки цих **трьох шарів** незалежними один від одного.

Щоб зробити документи і процеси правил управління незалежними один від одного, необхідно уникати того, щоб процес правил управління взаємодіяв з користувачами. Найбільшу перевагу треба надати тому, щоб робота шару управління БД була незалежна від правил управління. Отже, щоб зберегти шар управління БД незалежним, необхідно тримати правила процесів управління строго відокремленими від запитів БД і транзакцій.

По суті, у добре спроектованому додатку не існує шарів, які виконують будь-які функції, що належать до сфери відповідальності решти шарів. У цьому і полягає абстрагування. Залишаючись незалежним, кожен шар повинен приховувати конкретні деталі роботи від інших шарів. (Нагадаємо, що раціональна свідомість не може контролювати нервові імпульси або ритми серця).

Таблиця 2

Проект кожного шару в тришаровій архітектурі додатків

Шар додатка	Інтерфейс	Фокус проекта
Документ	GUI	Об'єкти, додатки, незалежні від правил
Процеси правил бізнесу	Процес	Інтерфейс користувача і вимоги до рішення, незалежні від даних
Управління БД	Транзакції і запити	Дані, незалежні від рішень

Як зазначалося, шар управління БД відповідає за забезпечення погодженості й захисту даних. З'ясуємо, як БД може підтримувати ці вимоги. Виявляється, що тільки за рахунок призначення доступу до конкретних даних через набір ретельно визначених транзакцій і запитів.

У інтерфейсі для шару управління БД транзакції мають відігравати три дуже важливі **ролі.**

Погодженість змін. Транзакції гарантують, що кожен запит або повністю завершено, або приймається припущення, що цього запиту взагалі не було. Наприклад, якщо гроші переміщуються з одного рахунку на інший, це переміщення або повністю закінчується, або не відбувається взагалі. У фінансових системах цей тип захисту запобігає "зникненню" коштів. Зрештою, проектування систем передбачає, що всі зміни відбуваються лише з допомогою транзакцій, які гарантують погодженість БД.

Забезпечення виконання основних правил управління. Крім погодженості, транзакції БД ретельно контролюють основні правила управлінської діяльності і забезпечують їх підтримку. Наприклад, будь-яка транзакція, що видаляє запис з БД, повинна спочатку перевірити, чи немає в цьому запису будь-яких некоректних дій щодо порушення правил управління. Якщо вони порушені, то код транзакції має відхилити даний запит, відмінити пов'язані з ним зміни БД і повідомити програмі в шарі правил управління, що ініційована транзакція не може стартувати.

Запобігання несанкціонованим або некоректним змінам у БД. У добре спроектованій системі транзакції є тільки способом внесення змін у БД. Інтерфейс транзакцій виконує функції сторожа, запобігаючи несанкціонованим або некоректним змінам у БД.

Усе, що належить до транзакцій інтерфейсу для шару управління БД, має гарантувати, що зміни внесені безпечно, секретно і погоджено. Але тільки внесення змін у БД для програм користувачів недостатньо. У користувачів мають бути можливості пошуку і вибору необхідної інформації. А це вже належить до функції запитів.

Аналогічно інтерфейсу транзакцій, що складається із сукупності спеціально спроектованих транзакцій, які коректно змінюють БД, інтерфейс запитів має включати до себе спеціально спроектовану множину запитів, які відбирають дані у шарі управління БД коректно і погоджено. Інколи ці запити називають логічними поданнями БД. Такі запити в інтерфейсі запитів мають обслуговувати три основні функції.

Спрощення складних з'єднань. Повний шар управління для великих організацій може мати кілька тисяч окремих файлів. Навіть у невеликих організаціях база даних у шарі управління БД може швидко зростати. Користувачі хочуть зрозуміти, як скомбінувати ці файли точно і правильно, щоб отримати відповіді на прості запитання. Завдання інтерфейсу запитів полягає в тому, щоб подати ці численні файли зведеними до дуже малої кількості простих, які містять у собі реальні дані. Виконуючи ці вимоги, інтерфейс запитів спрощує спілкування між користувачами і додатками і гарантує, що це спрощення зроблено точно і правильно.

Гарантування погодженості. Обов'язком шару запитів є гарантування того, щоб складні запити оброблялись коректно і виконували те, про що запитував додаток чи конкретний користувач.

Забезпечення секретності. Якщо запити сконструйовані належним чином, то вони мають гарантувати, що будь-яка інформація доступна тільки тим користувачам і додаткам, для яких передбачена можливість проглядання цієї інформації.

Інтерфейси запитів і транзакцій - це дві сторони однієї медалі. Один дає змогу проводити пошук інформації, інший - змінювати її. Разом ці дві частини інтерфейсу шару управління БД гарантують, що інформація в даному шарі залишається в погодженому стані, і тому

якщо користувачам буде потрібна якась інформація, **вони** обов'язково її отримають.

Взагалі, створюючи БД, незалежну від правил управління, необхідно дотримуватись чотирьох основних **вимог.**

Ретельно проектувати БД, використовуючи добре сплановану модель даних.

Розробляти запити і транзакції, що реалізують хорошу доступність БД. Бажано розробити "менеджера" транзакцій і запитів, **який** забезпечуватиме високорівневе подання даних. У разі змін "менеджер" транзакцій і запитів буде пакувати складні зміни у більші транзакції. З погляду вибірки попередньо запаковані запити дають змогу спрощено розглядати більш складну фізичну БД, яка міститься в основі.

Допускати тільки добре спроектовані транзакції для коригування БД.

Ізолювати користувачів від конкретних деталей і місцеположення БД, яка міститься в основі. Формальний інтерфейс, який використовується в шарі управління БД, має надавати додаткові переваги: додаток дістає здатність чітко взаємодіяти з БД багатьох комп'ютерів. Інакше кажучи, з допомогою коректно спроектованих формальних транзакцій і запитів інтерфейс БД може взаємодіяти **з** багатьма різними БД на різних комп'ютерах таким чином, щоб і користувач, і додаток не знали про це.

Підбиваючи підсумки, можна сформулювати переваги, що може надати тришарова архітектура додатків, у якій кожен шар додатку незалежний від інших шарів.

Більш стійкі БД. Якщо чітко відокремлювати правила управління від покладених

в основу даних, то цим зберігається незалежність БД від змін у політиці управління і підтримується погодженість і осмисленість даних протягом більш тривалого періоду.

Здатність до взаємодії. Відокремлюючи правила управління від інтерфейсу користувача і детальних коригувань БД, можливо зробити правила управління такими, що будуть повторно використовуватися. Здатність до взаємодії на рівні правил управління може зрештою стати дійсною.

Гнучкість і свобода. За дійсною незалежністю від покладених в основу правил управління додатки краще задовольнятимуть конкретні вимоги користувачів. Останні зможуть замовляти конкретний інтерфейс користувача, незважаючи на згадані правила управління. І навпаки, можна змінювати і розширювати процеси правил управління, не зачіпаючи інтерфейси користувача.

Розподіл зусиль. Побудова ефективних систем клієнт/сервер для великих організацій є значна за обсягом робота. Розробка якісних систем клієнт/сервер потребує знання і досвіду проектування графічного інтерфейсу користувача, програмування у сфері правил управління, БД, роботи в мережі, проектування розподіленої системи тощо. Кожному членові колективу розробників проекту знати усе це непотрібно, оскільки запропонована архітектура додатків забезпечує основу для розподілу робіт. Фактично тришарова архітектура додатків дає змогу зібрати колектив професіоналів комп'ютерної сфери з різних галузей знань, які мають відповідний досвід проектування і побудови системи, що є реальним.

Ключ до сприйняття тришарової архітектури - нагадування про те, що це архітектура на рівні логіки, а не на фізичному. Аналогічну тришарову архітектуру можна використовувати не тільки для розподілених, а й для централізованих систем. У цьому весь сенс: тришарова архітектура додатків - кращий засіб будувати додатки незалежно від того, розподілені вони чи ні. Така архітектура спрощує поділ компонентів додатку в разі потреби.

Ідея побудови ефективних розподілених додатків більш глибока, ніж просто з'ясування того, як розмістити фізичне обладнання. Тому ефективно спроектовані додатки можна розподіляти відносно легко, тоді як неефективні - практично неможливо. Концентрувати увагу на питанні "*що-йде-куди*" до того, як стане зрозумілим, яким чином будувати первинно якісні додатки, є помилкою. Тому потрібно брати до уваги ось що.

Добре спроектовані додатки, які побудовані навколо придатної архітектури додатків, можна розподіляти як завгодно.

Погано спроектовані додатки, у яких немає чіткої і погодженої архітектури додатків, розподілити неможливо, і, ймовірно, вони ніколи не працюватимуть у розподіленому середовищі; такі додатки не дуже добре працюють і в Централізованих системах.

- Ключем до ефективних розподілених систем є ефективне проектування додатків.
- Ефективні додатки працюватимуть і в централізованих, і в розподілених конфігураціях, і в будь-яких проміжних варіантах.
- Рішення про те, де працюватимуть визначені шари додатку, є вторинним.
- Використання зрозумілої і погодженої архітектури додатка є однією з центральних стратегій для розробок ефективних додатків і систем.

Людям, які працюють у сфері управління і програмних засобів і ретельно планують свою діяльність, тришарова архітектура в майбутніх проектах додатків може надати такі переваги:

- базу для побудови дуже гнучких додатків, які легко змінювати відповідно до

конкретних змін у політиці управління;

- високий рівень повторного використання програмних засобів;
- більш легку розробку великих складних додатків, які підтримують високу пропускну здатність як середовищ підтримки рішень, так і транзакцій, а також розподілених додатків, що підтримують централізоване управління і самокеровані колективи.