Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

CASE-ТЕХНОЛОГІЇ У ПРОЕКТУВАННІ ІНФОРМАЦІЙНИХ СИСТЕМ

НАВЧАЛЬНИЙ ПОСІБНИК

Кропивницький
Видавець Лисенко В.Ф.
2018

УДК 004.03(075.8)

Рекомендовано вченою радою ЦНТУ як навчальний посібник для студентів вищих навчальних закладів
Протокол № 9 від 29.05.2017 р.

Рецензенти:

Смірнов Олексій Анатолійович - д.т.н., проф., завідувач кафедри програмування та захисту інформації Центральноукраїнського національного технічного університету;
Паращук Степан Дмитрович - к.ф-м.н., доцент, в.о. завідувача кафедри інформатики Центральноукраїнського державного педагогічного університету.

Савеленко О.К., Лисенко І.А., Іванченко О.О.
С12 CASE-технології у проектуванні інформаційних систем: Навчальний посібник. - Кропивницький: Видавець Лисенко В.Ф., 2018.- 240 с.

У навчальному посібнику, складеному за навчальною програмою курсу «Технології проектування комп'ютерних систем», основну увагу приділено розкриттю тем курсу та вирішенню практичних завдань.

Призначений для студентів, що навчаються за спеціальністю: «Комп'ютерна інженерія», а також може буде корисним спеціалістам з автоматизації систем проектування друкованих плат.

УДК 004.03(075.8)

Передмова

Автоматизація проектування займає особливе місце серед інформаційних технологій.

Поява перших програм для автоматизації проектування відноситься до початку 60-х р.р. Тоді були створені програми для вирішення завдань будівельної механіки, аналізу електронних схем, проектування друкованих плат. Подальший розвиток САПР йшов:

- шляхом створення апаратних і програмних засобів машинної графіки;

- підвищення обчислювальної ефективності програм моделювання і аналізу;розширення сфер застосування САПР;

- спрощення призначеного для користувача інтерфейсу;

- впровадження в САПР елементів штучного інтелекту.

На сьогодення розроблено досить велике число програмно-методичних комплексів САПР з різними мірою спеціалізації і прикладної орієнтації. В результаті автоматизація проектування стала необхідною складовою частиною підготовки інженерів різних спеціальностей. Інженер, що не володіє знаннями і не уміє працювати з САПР, не може вважатися повноцінним фахівцем. Цей аспект пояснюється наступним:

- по-перше, автоматизація проектування — синтетична дисципліна, її складовими частинами є багато інших сучасних інформаційних технологій. Так, технічне забезпечення систем автоматизованого проектування (САПР) засноване на використанні обчислювальних мереж і телекомунікаційних технологій, в САПР використовуються персональні комп'ютери і робочі станції, є приклади застосування мейнфреймів. Математичне забезпечення

САПР відрізняється багатством і різноманітністю використовуваних методів обчислювальної математики, статистики, математичного програмування, дискретної математики, штучного інтелекту. Програмні комплекси САПР належать до найбільш складних сучасних програмних систем, заснованих на операційних системах Unix, Windows, мовах програмування С, С++, Java і інших сучасних CASE - технологіях проектування, реляційних і об'єктно-орієнтованих системах управління базами даних (СУБД), стандартах відкритих систем і обміну даними в комп'ютерних середовищах;

- по-друге, знання основ автоматизації проектування і уміння працювати із засобами САПР потрібно практично будь-якому інженерові-розробнику. Комп'ютерами насичені проектні підрозділи, конструкторські бюро і офіси. Робота конструктора за звичайним кульманом, розрахунки за допомогою логарифмічної лінійки або оформлення звіту на звичайній машинці стали анахронізмом. Підприємства та організації, що ведуть розробки без САПР або лише з малою мірою їх використання, виявляються неконкурентоздатними як із-за великих матеріальних і тимчасових витрат на проектування, так і через невисоку якість проектів завдяки впливу людського чинника.

Підготовка інженерів, володіючих методологією розробки САПР в області комп'ютерної інженерії, включає базову і спеціальну компоненти. Найбільш загальні стани, моделі і методики автоматизованого проектування входять в програму курсу, присвяченого основам САПР, детальніше вивчення методів і програм, які специфічні для конкретних спеціальностей, передбачається в профільних дисциплінах.

Дані відомості по різних аспектах і видах забезпечення систем автоматизованого проектування, необхідні кваліфікованим користувачам САПР в різних областях сфери діяльності людини. Значна увага приділена математичному забезпеченню процедур аналізу і синтезу проектних рішень, побудові локальних і корпоративних обчислювальних мереж САПР, складу і функціям системних середовищ САПР. Освітлені також методики концептуального проектування складних систем, які активно розвиваються останнім часом і покладені в основу технології CALS, а також питання інтеграції САПР з автоматизованими системами управління і діловодства.

Цей посібник орієнтований на базову підготовку студентів інженерних спеціальностей напряму підготовки «Комп'ютерна інженерія» в області розробки та експлуатації САПР.

Зміст

Вступ

Тенденції розвитку сучасних інформаційних технологій призводять до постійного зросту складності інформаційних систем (ІС), що створюються в різноманітних областях економіки. Сучасні великі проекти ІС характеризуються наступними особливостями:

- складністю опису (достатньо велика кількість функцій, процесів, елементів даних та складні взаємозв'язки між ними), що потребує досконалого моделювання і аналізу даних та процесів;

- наявністю сукупності тісно взаємодіючих компонентів (підсистем), що мають свої локальні задачі та цілі функціонування (наприклад, традиційних додатків, пов'язаних з обробкою транзакцій та розв'язанням регламентних задач та додатків аналітичної обробки (підтримки прийняття рішень), використовуючих нерегламентовані запити до даних великого обсягу);

- відсутністю прямих аналогів, що обмежує можливість використання будь-яких типових проектних рішень та прикладних систем;

- необхідністю інтеграції існуючих додатків та додатків, що розробляються заново;

- функціонуванням в неоднорідному середовищі на різноманітних апаратних платформах;

- розрізненістю та різнорідністю окремих груп розробників за рівнем кваліфікації та традиціями почерку проектувальної організації;

- суттєвою часовою протяжністю проекту, обумовленою, з одного боку, обмеженими можливостями колективу розробників, та

з іншого боку, масштабами організації-замовника і різноманітною ступеню готовності окремих її підрозділів по впровадженню ІС.

Для успішної реалізації проекту об'єкт проектування (ОП) має бути перш, за все, адекватно описано, мають бути побудовані повні та неконфліктні функціональні і інформаційні моделі ІС. Накопичений на сьогодення досвід проектування ІС показує, що це логічно складна, трудомістка задача, яка потребує високої кваліфікації спеціалістів. Однак до недавніх пір проектування ІС виконувалось в основному на інтуїтивному рівні з використанням неформалізованих методів з використанням досить великої частки проектування в інтерактивному режимі, заснованих на творчості, практичному досвіді, експертних оцінках та дорогих експериментальних перевірках якості функціонування ІС. Крім того, в процесі створення та функціонування ІС інформаційні потреби користувачів можуть змінюватися чи уточнятися, що ще більше ускладнює розробку та супроводження таких систем.

В 70-х та 80-х роках минулого століття при розробці ІС достатньо широко застосовувалась структурна методологія, що надає в розпорядження розробників суворі формалізовані методи опису ІС і технічних рішень. Вона заснована на наглядній графічній техніці: для опису різноманітного роду моделей ІС використовуються схеми та діаграми.

Наочність і суворість засобів структурного аналізу дозволяла розробникам та майбутнім користувачам системи з самого початку неформально брати участь в її створенні, обговорювати та закріпляти розуміння основних технічних рішень. Однак, широке розуміння цієї методології і використання її рекомендацій при розробці конкретних ІС зустрічалось достатньо рідко, оскільки при

неавтоматизованій (ручній або інтерактивній) розробці це практично неможливо. Дійсно, вручну дуже важко:

- розробити і графічно представити достовірні специфікації системи (принаймні - з першого разу);

- перевірити їх на повноту та неконфліктність;

- змінити конструкцію взагалі або її структурний модуль.

Якщо все ж вдасться створити комплект проектних документів, то його переробка при виявленні серйозних змін в конструкції ОП практично неможлива. Ручна розробка звичайно породжувала наступні проблеми:

- неадекватна специфікація вимог;

- нездатність виявляти помилки в проектних рішеннях;

- низька якість документації знижує експлуатаційні якості виробу в промисловій експлуатації;

- трудомісткий цикл тестування ОП в процесі дослідної експлуатації.

З іншого боку, розробники ІС історично завжди стояли останніми в ряді тих, хто використовував комп'ютерні технології для підвищення якості, надійності та продуктивності у своїй власній роботі (феномен "чоботаря без чобіт").

Перераховані фактори сприяли створенню програмно-технологічних засобів спеціального класу - CASE-засобів, що реалізують CASE-технологію проектування, виготовлення і супроводження ІС.

Термін CASE (Computer Aided Software Engineering) широко використовується в наш час. Первинне значення терміну CASE, обмежене питаннями автоматизації розробки лише програмного забезпечення (ПЗ), в наш час набуло нового сенсу і охоплює процес

розробки складних ІС в цілому. Тепер під терміном CASE-засоби розуміються програмні засоби, що підтримують процеси проектування, виготовлення і супроводження ІС, включаючи:

- аналіз та формулювання вимог;
- проектування прикладного ПЗ (додатків) та баз даних;
- генерацію коду;
- тестування;
- документування;
- забезпечення якості;
- конфігураційне керування та керування проектом, тощо.

CASE-засоби разом із системним ПЗ та технічними засобами створюють повне середовище розробки ІС.

Появленню CASE-технології та CASE-засобів передували дослідження в області методології програмування. Останнє придбало риси системного підходу з розробкою та впровадженням мов високого рівня, методів структурного та модульного програмування, мов проектування і засобів їх підтримки, формальних і неформальних мов опису системних вимог та специфікацій тощо. Крім того, появленню CASE-технології сприяли і такі фактори, як:

- підготовка аналітиків та програмістів на основі концепцій модульного та структурного програмування;
- широке впровадження і постійний ріст продуктивності комп'ютерів дозволили використовувати ефективні графічні засоби і автоматизувати більшість етапів проектування;
- впровадження мережевої технології забезпечує можливість об'єднання зусиль окремих виконавців в єдиний процес

проектування шляхом використання розподіленої бази даних, що містить необхідну інформацію про проект.

CASE-технологія являє собою методологію проектування ІС, а також набір інструментальних засобів, що дозволяють в наглядній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки та супроводження ІС, розробляти додатки у відповідності з інформаційними потребами користувачів. Більшість існуючих CASE-засобів заснована на методологіях структурного (в основному) чи об′єктно-орієнтованого аналізу та проектування, використовуючих специфікації у вигляді діаграм або текстів для опису зовнішніх вимог, зв’язків між моделями системи, динаміки поведінки системи і архітектури програмних засобів.

Згідно огляду передових технологій (Survey of Advanced Technology), що був представлений фірмою Systems Development Inc. у 2009 р. за результатами анкетування більше 1000 американських фірм, CASE-технологія в наш час потрапила в розряд найбільш стабільних інформаційних технологій (її використовувала половина всіх опитаних користувачів більш ніж в третині своїх проектів, із них 85% завершилися успішно). Однак, не зважаючи на всі потенційні можливості CASE-засобів, існує безліч прикладів їх невдалого впровадження, в результаті яких CASE-засоби стають "поличним" ПЗ (shelfware). У зв’язку з цим необхідно відзначити наступне:

- CASE-засоби не обов’язково дають негайний ефект зразу після впровадження виконаної розробки в промислову експлуатацію; він може бути отриманий лише через деякий час;

- реальні затрати на впровадження CASE-засобів звичайно набагато перевищують затрати на їх придбання;

- CASE-засоби забезпечують можливості для отримання суттєвої вигоди лише після успішного завершення процесу їх впровадження.

Через різноманітну природу CASE-засобів було б помилкою робити будь-які беззаперечні твердження відносно реального вдоволення тих чи інших очікувань від їх впровадження. Можна перелічити наступні фактори, що ускладнюють визначення можливого ефекту від використання CASE-засобів:

- широка різноманітність якостей і можливостей CASE-засобів;

- відносно невелика кількість часу використання CASE-засобів у різноманітних організаціях та недолік досвіду їх застосування;

- широка різноманітність у практиці впровадження організацій;

- відсутність детальних метрик і даних для вже виконаних і поточних проектів;

- широкий діапазон предметних областей проектів;

- різноманітний ступінь інтеграції CASE-засобів в проектах.

У результаті цих складностей інформація про реальні впровадження вкрай обмежена і заперечлива. Вона залежить від типу засобів, характеристик проектів, рівня супроводження та досвіду користувачів.

Аналітики вважають, що реальна вигода від використання деяких типів CASE-засобів може бути отримана лише після одно- чи дворічного періоду промислової експлуатації. Інші вважають, що вплив може реально виявитися лише у фазі експлуатації

життєвого циклу ІС, коли технологічні покращення можуть призвести до зниження експлуатаційних витрат.

Для успішного впровадження CASE-засобів організація має, як мінімум, володіти наступними якостями:

- технологія: розуміння обмеженості існуючих можливостей та здатність прийняти нову технологію;

- культура: готовність до впровадження нових процесів і взаємовідносин між розробниками і користувачами;

- керування: чітке керівництво і організованість по відношенню до найбільш важливих етапів і процесів впровадження.

Якщо організація не володіє хоча б одним з перелічених якостей, то впровадження CASE-засобів може закінчитися невдачею незалежно від ретельності дотримання різноманітних рекомендацій по впровадженню.

Для того, щоб прийняти вагоме рішення відносно інвестицій в CASE-технологію, користувачі вимушені проводити оцінку окремих CASE-засобів, спираючись на неповні та протилежні дані. Ця проблема найчастіше ускладнюється недостатніми знаннями всіх можливих "підводних каменів" використання CASE-засобів. Серед найбільш важливих проблем виділимо наступні:

- достовірна оцінка віддачі від інвестицій в CASE-засоби є дещо ускладненим процесом через відсутність прийнятних метрик і даних з проектів та процесів розробки ПЗ (часткова невизначеність даних);

- впровадження CASE-засобів може бути досить тривалим процесом і тому не принести негайної віддачі. Можливе навіть тимчасове зниження продуктивності внаслідок зусиль, витрачених на впровадження. Внаслідок цього керівництво організації-

користувача може втратити цікавість до CASE-засобів, віднести їх до розряду неперспективних технологій і призупинити підтримку їх впровадження;

- відсутність повної відповідності між тими процесами і методами, які підтримуються CASE-засобами, і тими, які використовуються в даній організації, може призвести до виникнення додаткових труднощів;

- CASE-засоби часто важко використовувати в комплексі з іншими подібними засобами. Це пояснюється як різними парадигмами, підтримуваними різними засобами, так і проблемами передачі даних і управління від одного засобу до іншого;

- деякі CASE-засоби вимагають надто багато зусиль для того, щоб виправдати їх використання в невеликому проекті, при цьому, проте, можна отримати вигоду з ОП, до якого зобов'язує його застосування;

- негативне ставлення персоналу до впровадження нової CASE-технології може бути головною причиною провалу проекту.

Користувачі CASE-засобів повинні бути готові до необхідності довготермінових витрат на експлуатацію, частої появи нових версій і можливого швидкого морального старіння засобів, а також постійних витрат на навчання і підвищення кваліфікації персоналу.

Незважаючи на всі висловлені застереження і деякий песимізм, грамотний і розумний підхід до використання CASE-засобів може подолати всі перераховані труднощі. Успішне впровадження CASE-засобів забезпечує одержання наступних вигод:

- високого рівня технологічної підтримки процесів розробки та супроводу ПЗ;

- позитивного впливу на деякі або всі з перелічених факторів: продуктивність, якість продукції, дотримання стандартів, документування;

- прийнятного рівня віддачі від інвестицій в CASE-засоби.

1. Основи методології проектування ІС

1.1. Життєвий цикл ІС

Одним з базових понять методології проектування ІС є поняття життєвого циклу її програмного забезпечення (ЖЦ ПЗ). ЖЦ ПЗ - це безперервний процес, який починається з моменту прийняття рішення про необхідність його створення і закінчується в момент його повного вилучення з експлуатації.

Основним нормативним документом, який регламентує ЖЦ ПЗ, є міжнародний стандарт ISO/IEC 12207 (ISO - International Organization of Standardization - Міжнародна організація по стандартизації, IEC - International Electrotechnical Commission - Міжнародна комісія з електротехніки). Він визначає структуру ЖЦ, що містить процеси, дії і задачі, які повинні бути вирішенні під час розробки ПЗ.

Структура ЖЦ ПЗ за стандартом ISO/IEC 12207 базується на трьох групах процесів:

- основні процеси ЖЦ ПЗ (придбання, постачання, розробка, експлуатація, супровід);

- допоміжні процеси, що забезпечують виконання основних процесів (документування, управління конфігурацією, забезпечення якості, верифікація, атестація, оцінка, аудит, вирішення поточно-виникаючих проблем);

- організаційні процеси (управління проектами, створення інфраструктури проекту, визначення, оцінка та поліпшення самого ЖЦ, навчання).

Розробка включає в себе всі роботи зі створення ПЗ і його компонент відповідно до заданих вимог, включаючи:

- оформлення проектної та експлуатаційної документації;
- підготовку матеріалів, необхідних для перевірки працездатності і відповідної якості програмних продуктів, матеріалів, необхідних для організації навчання персоналу тощо.

Розробка ПЗ включає в себе, як правило, аналіз, проектування і реалізацію.

Експлуатація включає в себе роботи з впровадження компонентів ПЗ в експлуатацію, в тому числі:

- конфігурування бази даних і робочих місць користувачів;
- забезпечення експлуатаційної документації;
- проведення навчання персоналу і безпосередньо експлуатацію;
- локалізацію проблем та усунення причин їх виникнення;
- модифікацію ПЗ в рамках встановленого регламенту;
- підготовку пропозицій щодо вдосконалення, розвитку та модернізації системи.

Управління проектом пов’язане з питаннями планування та організації робіт, створення колективів розробників і контролю за термінами та якістю виконуваних робіт. Технічне та організаційне забезпечення проекту включає:

- вибір методів і інструментальних засобів для реалізації проекту;
- визначення методів опису проміжних станів розробки;
- розробку методів і засобів випробувань ПЗ;
- навчання персоналу тощо.

Забезпечення якості проекту пов’язане з проблемами верифікації, перевірки та тестування ПЗ.

Верифікація - це процес визначення того, чи відповідає поточний стан розробки, досягнутий на даному етапі, вимогам цього етапу.

Перевірка дозволяє оцінити відповідність параметрів розробки вихідним вимогам. Перевірка частково збігається з тестуванням, яке пов’язане з ідентифікацією відмінностей між дійсними і очікуваними результатами та оцінкою відповідності характеристик ПЗ вихідним вимогам.

В процесі реалізації проекту важливе місце займають питання ідентифікації, опису і контролю конфігурації окремих компонентів і всієї системи в цілому.

Управління конфігурацією є одним з допоміжних процесів, які підтримують основні процеси життєвого циклу ПЗ, перш за все - процеси розробки та супроводу ПЗ. При створенні проектів складних ІС, до архітектури яких входить багато компонентів, кожен з яких може мати різновиди або версії, виникає проблема обліку їх зв’язків та функцій, створення уніфікованої структури та забезпечення розвитку всієї системи. Управління конфігурацією дозволяє організувати, систематично враховувати і контролювати внесення змін до ПЗ на всіх стадіях ЖЦ. Загальні принципи і рекомендації конфігураційного обліку, планування та управління конфігураціями ПЗ відображені в проекті стандарту ISO 12207-2 [5].

Кожен процес характеризується:

- певними задачами і методами їх вирішення;
- вихідними даними, отриманими на попередньому етапі;

- результатами.

Результатами аналізу, зокрема, є функціональні моделі, інформаційні моделі і відповідні їм діаграми. ЖЦ ПЗ носить ітераційний характер: результати чергового етапу часто викликають зміни в проектних рішеннях, вироблених на більш ранніх етапах розробки.

1.2. Моделі життєвого циклу ПЗ

Стандарт ISO/IEC 12207 не пропонує конкретну модель ЖЦ і методи розробки ПЗ (під моделлю ЖЦ розуміється структура, що визначає послідовність виконання та взаємозв'язку процесів, дій і завдань, що виконуються протягом ЖЦ. Модель ЖЦ залежить від специфіки ІС та специфіки умов, в яких остання створюється і функціонує). Його регламенти є загальними для будь-яких моделей ЖЦ, методологій і технологій розробки. Стандарт ISO/IEC 12207 описує структуру процесів ЖЦ ПЗ, але не конкретизує в деталях, як реалізувати або виконати дії і задачі, включені в ці процеси.

На сьогодення найбільшого поширення набули наступні дві основні моделі ЖЦ:

- каскадна модель (1970-1985 р.р.);
- спіральна модель (1986-1990 р.р.).

Спочатку існували однорідні ІС, кожен додаток яких являв собою єдине ціле. Для розробки такого типу додатків застосовувався каскадний спосіб. Його основною характеристикою є розбиття всієї розробки на етапи, причому перехід з одного етапу на наступний відбувається тільки після того, як буде повністю завершена робота на поточному (рисунок 1.2.1). Кожен етап

завершується випуском повного комплекту документації, достатнього для того, щоб розробка могла бути продовжена іншою командою розробників.

Позитивні сторони застосування каскадного підходу полягають в наступному [2]:

- на кожному етапі формується закінчений набір проектної документації, який відповідає критеріям повноти та узгодженості;
- виконуючись в логічній послідовності, етапи робіт дозволяють планувати терміни завершення всіх робіт і відповідні витрати.

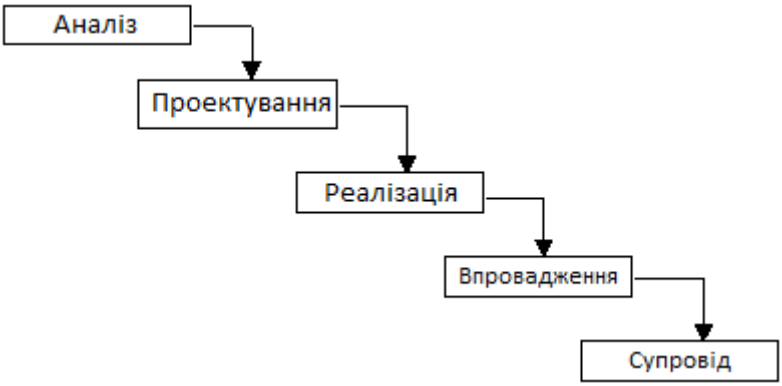

Рисунок 1.2.1 -- Каскадна схема розробки ПЗ

Каскадний підхід добре зарекомендував себе при побудові ІС, для яких на самому початку розробки можна досить точно і повно сформулювати всі вимоги, з тим, щоб надати розробникам свободу їх найкращої реалізації з технічної точки зору. У цю

категорію потрапляють складні розрахункові системи, системи реального часу і інші подібні задачі.

Однак, в процесі використання цього підходу, виявився ряд його недоліків, спричинених насамперед тим, що реальний процес створення ПЗ ніколи повністю не вкладався в таку жорстку схему. У процесі створення ПЗ постійно виникала потреба у поверненні до попередніх етапів для уточнення або перегляду раніше прийнятих рішень. У результаті реальний процес створення ПЗ набрав такого вигляду (рисунок 1.2.2):

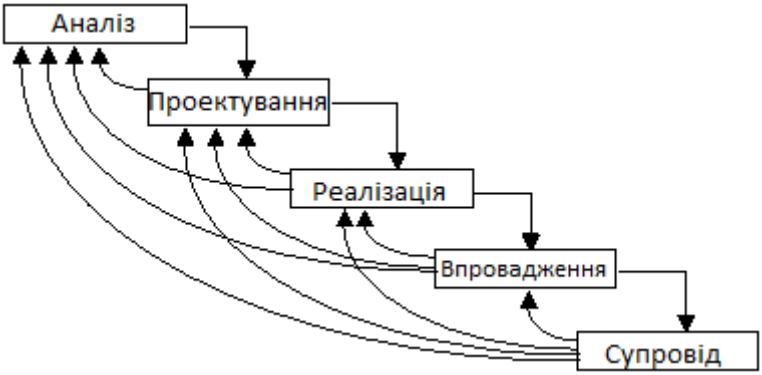

Рисунок 1.2.2 - Реальный процес розробки ПЗ за каскадною схемою

Основним недоліком каскадного підходу є суттєве запізнення з отриманням результатів. Узгодження результатів з користувачами проводиться тільки в точках, що плануються після завершення кожного етапу робіт, вимоги до ІС "заморожені" у вигляді технічного завдання на весь час її створення. Таким чином,

користувачі можуть внести свої зауваження тільки після того, як робота над системою буде повністю завершена.

У разі неточного викладу вимог або їх зміни протягом тривалого періоду створення ПЗ, користувачі отримують систему, яка не відповідає їх потребам. Моделі (як функціональні, так і інформаційні) автоматизації об’єкта можуть застаріти одночасно з їх затвердженням.

Для подолання перелічених проблем була запропонована спіральна модель ЖЦ [2] (рисунок 1.2.3), в основу якої покладена робота з початковими етапами ЖЦ: аналіз та проектування. На цих етапах реалізація технічних рішень перевіряється шляхом створення прототипів. Кожен виток спіралі відповідає створенню фрагмента або версії ПЗ, на ньому уточнюються цілі і характеристики проекту, визначається його якість і плануються роботи наступного витка спіралі. Так поглиблюються і послідовно конкретизуються деталі проекту і в результаті вибирається обґрунтовано кращий варіант, який доводиться до реалізації.

Розробка ітераціями відображає об’єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При інтерактивному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання - якнайшвидше показати майбутнім користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог.

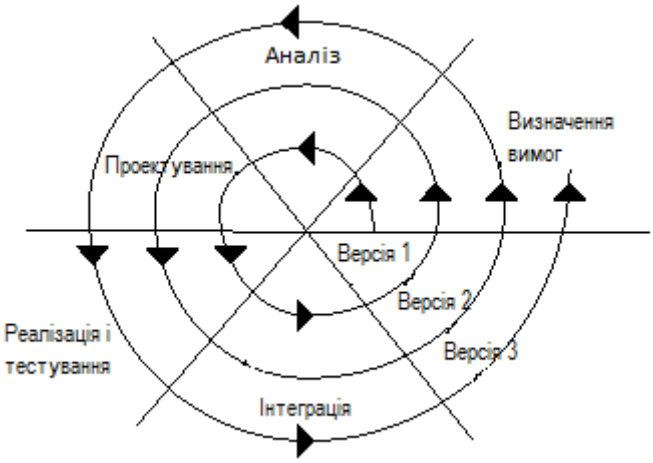

Рисунок 1.2.3 - Спіральна модель ЖЦ

Основна проблема спірального циклу - визначення моменту переходу на наступний етап. Для її вирішення необхідно ввести тимчасові обмеження на кожен з етапів життєвого циклу. Перехід здійснюється відповідно до плану, навіть якщо не вся запланована робота закінчена. План складається на основі статистичних даних, отриманих в попередніх проектах, і особистого досвіду розробників, які мають високу кваліфікацію в даній сфері розробки проекту, від особистого почерку головного конструктора організації - розробника, від рівня кваліфікації постановників задач тощо.

1.3. Методології та технології проектування ІС

1.3.1. Загальні вимоги до методологій та технологій проектування ІС

Методології, технології та інструментальні засоби проектування (CASE-засоби) складають основу проекту будь-якої ІС. Методологія реалізується через конкретні технології і підтримує їх стандарти, методики та інструментальні засоби, які забезпечують виконання процесів ЖЦ.

Технологія проектування визначається, як сукупність трьох складових:

- покрокової процедури, яка визначає послідовність технологічних операцій проектування (рисунок 1.3.1.1);
- критеріїв і правил, що використовуються для оцінки результатів виконання технологічних операцій;
- нотацій (графічних і текстових засобів), що використовуються для опису системи, яка підлягає проектуванню.

Технологічні інструкції, компоненти основного змісту технології повинні складатися з опису послідовності технологічних операцій, умов, залежно від яких виконується та чи інша операція, і описів самих операцій.

Технологія проектування, розробки та супроводу ІС повинна відповідати наступним загальним вимогам:

- технологія повинна підтримувати повний ЖЦ ПЗ;
- технологія повинна забезпечувати гарантоване досягнення цілей розробки ІС з заданою якістю і у встановлений час;

- технологія повинна забезпечувати можливість виконання великих проектів у вигляді підсистем (тобто, можливість декомпозиції проекту на складові частини, що розробляються групами виконавців обмеженої чисельності з подальшою інтеграцією складових частин в єдиний проект).

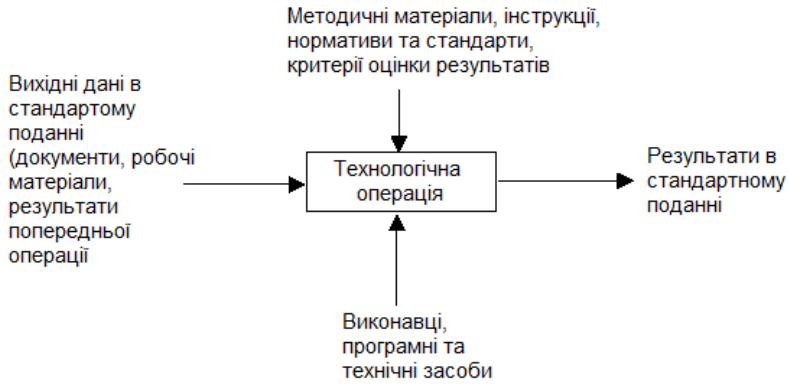

Рисунок 1.3.1.1 - Структура технологічної операції проектування

Досвід розробки великих ІС показує, що для підвищення ефективності проектних робіт необхідно розбити проект на окремі підсистеми, використавши стандартну методику декомпозиції. Тоді стає можливою проектування окремих підсистем/модулів окремими групами фахівців. При цьому необхідно забезпечити:

- координацію ведення спільного проекту;

- виключення дублювання результатів робіт кожної проектної групи, що може виникнути в силу наявності загальних вхідних та проміжних даних і функцій та звичного значного відсотку людського фактору;

- можливість ведення робіт з проектування окремих підсистем/модулів невеликими групами (3-7 чоловік). Це обумовлено принципами керованості колективу і підвищенням продуктивності за рахунок мінімізації числа зовнішніх зв'язків;

- мінімальний час отримання працездатної ІС. Мова йде не про термін готовності всієї розробки (ІС), а про терміни реалізації окремих підсистем/модулів окремими групами розробників. Реалізація ІС в цілому в короткі терміни може зажадати залучення великої кількості розробників, при цьому ефект може виявитися нижчим, ніж при реалізації в більший термін часу окремих підсистем меншою кількістю розробників. Практика показує, що навіть за наявності повністю завершеного проекту, впровадження йде послідовно по окремих підсистемах, забезпечуючи:

- можливість керування конфігурацією проекту;

- можливість ведення версій проекту і його складових;

- можливість автоматичного випуску проектної документації та синхронізацію її версій з версіями проекту;

- незалежність виконуваних проектних рішень від засобів реалізації ІС ((систем управління базами даних (СУБД), операційних систем, мов і систем програмування));

- організацію підтримки проекту комплексом узгоджених CASE-засобів, що забезпечують автоматизацію процесів і виконуються на всіх стадіях ЖЦ ОП. Загальний підхід до оцінки та вибору CASE-засобів описаний у розділі 4, приклади комплексів CASE-засобів - у підрозділі 5.7.

Реальне застосування будь-якої технології проектування, розробки та супроводження ІС в конкретній організації і конкретному проекті неможливе без розробки ряду стандартів

(правил, угод), які повинні дотримуватися всіма учасниками проекту. До таких стандартів належать наступні:

- стандарт проектування;
- стандарт оформлення проектної документації;
- стандарт користувача інтерфейсу.

Стандарт проектування повинен встановлювати:

- набір необхідних моделей (діаграм) на кожному етапі проектування та ступінь їх деталізації;
- правила фіксації проектних рішень на діаграмах, в тому числі: правила іменування об'єктів (включаючи угоди за термінологією), набір атрибутів для всіх об'єктів та правила їх заповнення на кожній стадії, правила оформлення діаграм, включаючи вимоги до форми і розмірів об'єктів тощо.;
- вимоги до конфігурації робочих місць розробників, включаючи налаштування операційної системи, CASE-засобів, загальні налаштування проекту тощо;
- механізм забезпечення спільної роботи над проектом, у тому числі: правила інтеграції підсистем проекту, правила підтримки проекту в однаковому для всіх розробників стані (регламент обміну проектною інформацією, механізм фіксації загальних об'єктів тощо), правила перевірки проектних рішень на несуперечність і т. д.

Стандарт оформлення проектної документації повинен встановлювати:

- комплектність, склад і структуру документації на кожній стадії проектування;
- вимоги до її оформлення (включаючи вимоги до змісту розділів, підрозділів, пунктів, таблиць тощо);

- правила підготовки, розгляду, погодження та затвердження документації із зазначенням граничних термінів для кожної стадії;

- вимоги до налаштування видавничої системи, що використовується в якості вбудованого засобу підготовки документації;

- вимоги до налаштування CASE-засобів для забезпечення підготовки документації відповідно до встановлених вимог.

Стандарт інтерфейсу користувача повинен встановлювати:

- правила оформлення екранів (шрифти і кольорова палітра), склад і розташування вікон та елементів керування;

- правила використання клавіатури і миші;

- правила оформлення текстів допомоги;

- перелік стандартних повідомлень;

- правила обробки реакції користувача.

1.3.2. Методологія RAD

Одним з можливих підходів до розробки ПЗ в рамках спіральної моделі ЖЦ, що набула останнім часом широкого поширення, є методологія швидкої розробки додатків RAD (Rapid Application Development). Під цим терміном звично розуміється процес розробки ПЗ, що містить 3 елементи:

- невелику команду програмістів (від 2 до 10 осіб);

- короткий, але ретельно пророблений виробничий графік (від 2 до 6 міс);

- повторюваний цикл, при якому розробники, у міру того, як додаток починає набувати форму, запитують і реалізують в продукті вимоги, отримані через взаємодію з замовником.

Команда розробників повинна являти собою групу професіоналів високої кваліфікації, що мають досвід в аналізі, проектуванні, генерації програмного коду і тестуванні ПЗ з використанням CASE-засобів. Члени колективу повинні також вміти трансформувати в робочі прототипи пропозиції кінцевих користувачів.

Життєвий цикл ПЗ за методологією RAD складається з чотирьох фаз:

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

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

На фазі проектування частина користувачів приймає участь у технічному проектуванні системи під керівництвом фахівців-розробників. CASE-засоби використовуються для швидкого отримання працюючого прототипу. Користувачі, безпосередньо

взаємодіючи з ними, уточнюють і доповнюють вимоги до системи, які не були виявлені на попередній фазі. Більш ретельно розглядаються процеси системи. Аналізується і, при необхідності, корегується функціональна модель. Кожен процес розглядається детально. При необхідності для кожного елементарного процесу створюється частковий прототип: екран, діалог, звіт, що усуває неясності або неоднозначності. Визначаються вимоги розмежування доступу до даних. На цій же фазі відбувається визначення набору необхідної документації.

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

- загальна інформаційна модель системи;

- функціональні моделі системи в цілому і підсистеми, що реалізуються окремими командами розробників;

- точно визначені за допомогою CASE-засобу інтерфейси між підсистемами, що автономно розробляються;

- побудовані прототипи екранів, звітів, діалогів.

Усі моделі та прототипи повинні бути отримані із застосуванням тих CASE-засобів, які будуть використовуватися в подальшому при побудові системи. Дана вимога викликана тим, що в традиційному підході при передачі інформації про проект з етапу на етап може відбутися фактично неконтрольоване спотворення

даних. Застосування єдиного середовища зберігання інформації про проект дозволяє уникнути цієї небезпеки.

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

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

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

- визначається необхідність розподілу даних;

- проводиться аналіз використання даних;

- проводиться фізичне проектування бази даних;

- визначаються вимоги до апаратних ресурсів;

- визначаються способи збільшення продуктивності;

- завершується розробка конструкторської і програмної документації проекту.

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

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

Наведена схема розробки ІС не є абсолютною. Можливі різні варіанти, залежні, наприклад, від початкових умов, в яких ведеться розробка:

- розробляється зовсім нова система;

- вже було проведено обстеження підприємства і існує модель його діяльності, яка може бути використана в якості основи для нової розробки;

- на підприємстві вже існує деяка ІС, яка може бути використана в якості початкового прототипу або повинна бути інтегрована з тією, що розробляється.

Слід, однак, відзначити, що методологія RAD, як і будь-яка інша, не може претендувати на універсальність, вона зручна, в першу чергу, для відносно невеликих проектів, що розробляються для конкретного замовника. Якщо ж розробляється типова система,

яка не є закінченим продуктом, а являє собою комплекс типових компонентів, що централізовано адаптуються до:

- програмно-технічних платформ;
- СУБД;
- засобів телекомунікації;
- організаційно-економічних особливостей об'єктів впровадження;
- та інтегруються з існуючими розробками,

на перший план виступають такі показники проекту, як керованість і якість, які можуть увійти в конфлікт з простотою і швидкістю розробки. Для таких проектів необхідні високий рівень планування і жорстка дисципліна проектування, суворе дотримання заздалегідь розроблених протоколів і інтерфейсів, що знижує швидкість розробки.

Методологія RAD непридатна для побудови складних розрахункових програм, операційних систем або програм керування космічними кораблями, тобто програм, що вимагають написання великого об'єму (сотні тисяч рядків) унікального програмного коду.

Не підходять для розробки за методологією RAD програми, в яких:

- відсутня яскраво виражена інтерфейсна частина;
- наочно визначена логіка роботи системи (наприклад, програма реального часу) і програми, від яких залежить безпека людей (наприклад, керування літаком або атомною електростанцією), так як ітеративний підхід припускає, що перші кілька версій напевно не будуть повністю працездатні, що в даному випадку виключається.

Оцінка розміру програм проводиться на основі функціональних елементів (екрани, повідомлення, звіти, файли і т.д.) Подібна метрика не залежить від мови програмування, на якій ведеться розробка.

Як підсумок, перерахуємо основні принципи методології RAD:

- розробка програм ітераціями;

- необов'язковість повного завершення робіт на кожному з етапів життєвого циклу;

- обов'язкове залучення користувачів до процесу розробки ІС;

- необхідне застосування CASE-засобів, що забезпечують цілісність проекту;

- застосування засобів керування конфігурацією для полегшення внесення змін у проект і супровід готової системи;

- обов'язкове використання генераторів коду;

- використання прототипування для більш повного з'ясування і задоволення потреб кінцевого користувача;

- тестування і розвиток проекту, які здійснюються одночасно з розробкою;

- ведення розробки нечисленною, добре керованою командою професіоналів;

- грамотне керівництво розробкою системи, чітке планування і контроль виконання робіт упродовж всієї роботи над виконанням проекту.

2. Структурний підхід до проектування ІС

2.1. Сутність структурного підходу

Сутність структурного підходу до розробки ІС полягає в її декомпозиції (розбивання) на функціональні підсистеми, які в свою чергу діляться на підфункції, що підрозділяються на завдання і так далі. Процес розбиття триває аж до конкретних процедур. При цьому система, що автоматизується, зберігає цілісне уявлення, в якому всі складові компоненти взаємопов'язані. При розробці системи "знизу-догори" від окремих завдань до всієї системи цілісність втрачається, виникають проблеми при інформаційному стикуванні окремих компонентів.

Усі найбільш поширені методології структурного підходу [8] базуються на ряді загальних принципів [3]. В якості найбільш поширених використовуються наступні два:

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

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

- принцип абстрагування - полягає у виділенні істотних аспектів системи і виключення несуттєвих для даного проекту;

- принцип формалізації - полягає в необхідності суворого методичного підходу до вирішення проблеми;

- принцип несуперечності - полягає в обґрунтування та узгодження окремих елементів проекту;

- принцип структурування даних - полягає в тому, що дані повинні бути структуровані і ієрархічно організовані.

У структурному аналізі використовуються в основному дві групи засобів, що ілюструють функції, які виконуються системою, і відносини між даними. Кожній групі засобів відповідають певні види моделей (діаграм), найбільш поширеними серед яких є наступні:

- SADT (Structured Analysis and Design Technique) моделі і відповідні функціональні діаграми (підрозділ 2.2);

- DFD (Data Flow Diagrams) діаграми потоків даних (підрозділ 2.3);

- ERD (Entity-Relationship Diagrams) діаграми "сутність-зв’язок" (підрозділ 2.4).

На стадії проектування ІС моделі розширюються, уточнюються і доповнюються діаграмами, що відображають структуру програмного забезпечення:

- архітектуру ПЗ;

- структурні схеми програм;

- діаграми екранних форм.

Перераховані моделі в сукупності дають повний опис ІС незалежно від того, чи є вона існуючою чи знову розробляється. Склад діаграм в кожному конкретному випадку залежить від необхідної повноти опису системи.

2.2. Методологія функціонального моделювання SADT

Методологія SADT розроблена Дугласом Россом і отримала подальший розвиток в роботі [4]. На її основі розроблена, зокрема, відома методологія IDEF0 (Icam DEFinition), яка є основною частиною програми ICAM (Інтеграція комп'ютерних та промислових технологій), що проводиться за ініціативою ВПС США.

Методологія SADT являє собою сукупність методів, правил і процедур, призначених для побудови функціональної моделі об'єкта будь-якої предметної області.

Функціональна модель SADT відображає функціональну структуру об'єкта, тобто - вироблені ним дії та зв'язки між цими діями. Основні елементи цієї методології ґрунтуються на наступних концепціях:

- графічне представлення блокового моделювання. Графіка блоків і дуг SADT-діаграми відображає функцію у вигляді блоку, а інтерфейси входу/виходу представляються дугами, які відповідно входять у блок і виходять з нього. Взаємодія блоків один з одним описуються у вигляді інтерфейсних дуг і виражають "обмеження", які, в свою чергу, визначають, коли і яким чином функції виконуються і керуються;

- суворість і точність. Виконання правил SADT вимагає достатньої суворості й точності, не накладаючи в той же час надмірних обмежень на дії аналітика.

Правила SADT включають:

- обмеження кількості блоків на кожному рівні декомпозиції (правило 3-6 блоків);

- зв’язок діаграм (номери блоків);

- унікальність міток і найменувань (відсутність імен, що повторюються);

- синтаксичні правила для графіки (блоки і дуги);

- поділ входів та керувань (правило визначення ролі даних);

- відокремлення організації від функції, тобто - виключення впливу організаційної структури на функціональну модель.

Методологія SADT може використовуватися для моделювання широкого кола систем і визначення вимог і функцій, а потім для розробки системи, яка задовольняє цим вимогам і реалізує ці функції. Для вже існуючих систем SADT може бути використана для аналізу функцій, які виконуються системою, а також для вказівки механізмів, за допомогою яких вони здійснюються.

2.2.1. Склад функціональної моделі

Результатом застосування методології SADT є модель, яка складається з діаграм, фрагментів текстів і глосаріїв, що мають посилання один на одного.

Діаграми - головні компоненти моделі, всі функції ІС та інтерфейси на них представлені як блоки і дуги. Місце з’єднання дуги з блоком визначає тип інтерфейсу.

Керуюча інформація входить в блок угорі, в той час, як інформація, що піддається обробці, показана з лівого боку блоку, а результати виходу показані з правого боку.

Механізм виконання (людина або автоматизована система), що здійснює операцію, представляється дугою, яка входить в блок знизу (рисунок 2.2.1.1).

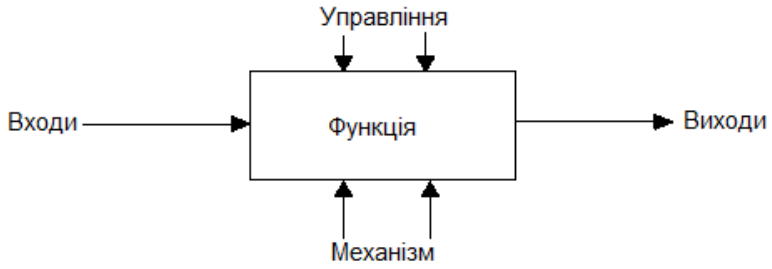

Рисунок 2.2.1.1 - Функціональний блок та інтерфейсні дуги

Однією з найбільш важливих особливостей методології SADT є поступове введення все більших рівнів деталізації у міру створення діаграм, що відображають модель.

На рисунку 2.2.1.2 наведені чотири діаграми та їх взаємозв'язок, показана структура SADT-моделі. Кожен компонент моделі може бути декомпозовано на іншій діаграмі. Кожна діаграма ілюструє "внутрішню будову" блоку на батьківській діаграмі.

2.2.2. Ієрархія діаграм

Побудова SADT-моделі починається з представлення всієї системи у вигляді найпростішої компоненти - одного блоку і дуг, що зображують інтерфейси з функціями поза системою. Оскільки єдиний блок представляє всю систему як єдине ціле, ім'я, вказане в блоці, є спільним. Це справедливо і для інтерфейсних дуг - вони

також представляють повний набір зовнішніх інтерфейсів системи в цілому.

Потім блок, який представляє систему в якості єдиного модуля, деталізується на іншій діаграмі за допомогою декількох блоків, з'єднаних інтерфейсними дугами. Ці блоки представляють основні підфункції вихідного блоку, межі якого визначені інтерфейсними дугами. Кожна з цих підфункцій може бути декомпозована подібним чином для більш детального уявлення.

У всіх випадках кожна підфункція може містити тільки ті елементи, які входять у вихідну функцію (рисунок 2.2.2.1). Крім того, з моделі не можна видаляти будь-які елементи, тобто, як уже зазначалося, батьківський блок і його інтерфейси забезпечують контекст. До нього не можна нічого додавати, і з нього не можна нічого видаляти.

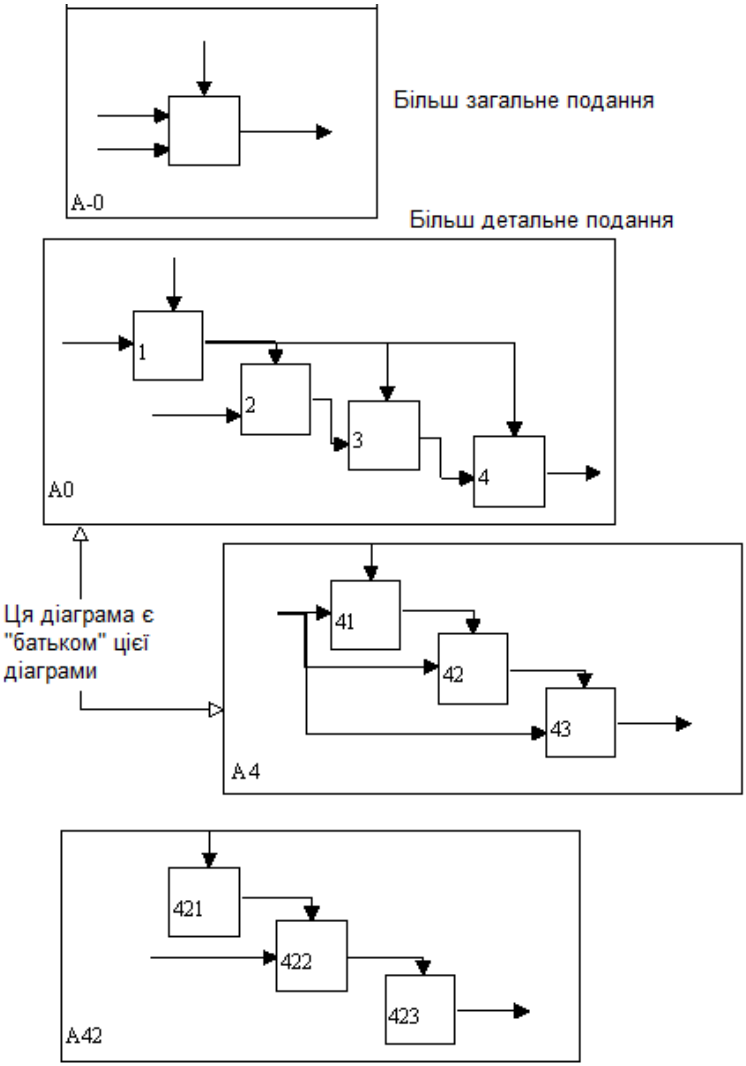

Рисунок 2.2.2.1 - Структура SADT-моделі. Декомпозиція діаграм

Модель SADT являє собою серію діаграм із супровідною документацією, які розбивають складний об’єкт на складові частини, представлені у вигляді блоків. Деталі кожного з основних блоків показані в вигляді блоків на інших діаграмах. Кожна детальна діаграма є декомпозицією блоку з більш загальної діаграми вищого рівня. На кожному кроці декомпозиції більш загальна діаграма називається батьківською для більш детальної діаграми нижчого рівня.

Дуги, що входять в блок і виходять з нього на діаграмі верхнього рівня, є точно тими ж самими, що й дуги, що входять до діаграми нижнього рівня і виходять з неї, тому що блок і діаграма представляють одну і ту ж частину системи.

На рисунках 2.2.2.2 - 2.2.2.4 представлені різні варіанти виконання функцій і з’єднання дуг з блоками, які найбільш використовуються розробниками ІС та ОП.

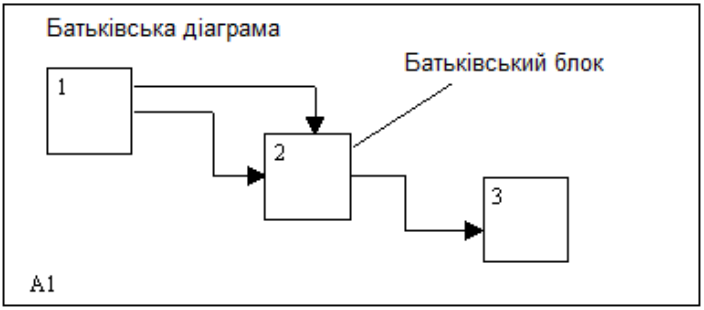

Рисунок 2.2.2.2 - Одночасне виконання

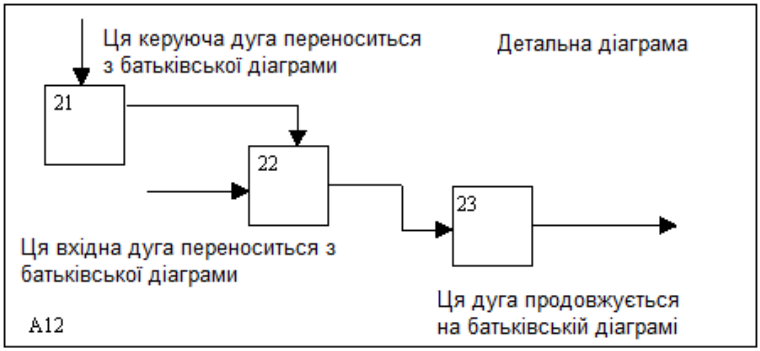

Рисунок 2.2.2.3 - Відповідність має бути повною і несуперечливою

Деякі дуги приєднані до блоків діаграми обома кінцями, в інших же один кінець залишається не приєднаним. Не приєднанні дуги відповідають входам, механізмам керуванням і виходам батьківського блоку. Джерело або одержувач цих граничних дуг можуть бути виявленими тільки на батьківській діаграмі. Не приєднанні кінці повинні відповідати дугам на вихідній діаграмі. Всі граничні дуги повинні продовжуватися на батьківській діаграмі, щоб вона була повною і несуперечливою.

На SADT-діаграмах не вказані явно ні послідовність, ні час. Зворотні зв'язки, ітерації, триваючі процеси і функції, що перекриваються (за часом), можуть бути зображені за допомогою дуг. Зворотні зв'язки можуть виступати у вигляді коментарів, зауважень, виправлень тощо (рисунок 2.2.2.4).

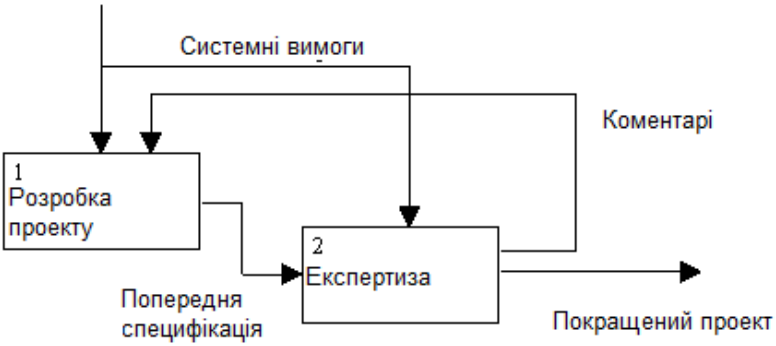

Рисунок 2.2.2.4 - Приклад зворотного зв’язку

Як було відзначено, механізми (дуги з нижнього боку) показують засоби, за допомогою яких здійснюється виконання функцій. Механізм може бути людиною, комп’ютером або будь-яким іншим пристроєм, який допомагає виконувати цю функцію (рисунок 2.2.2.5).

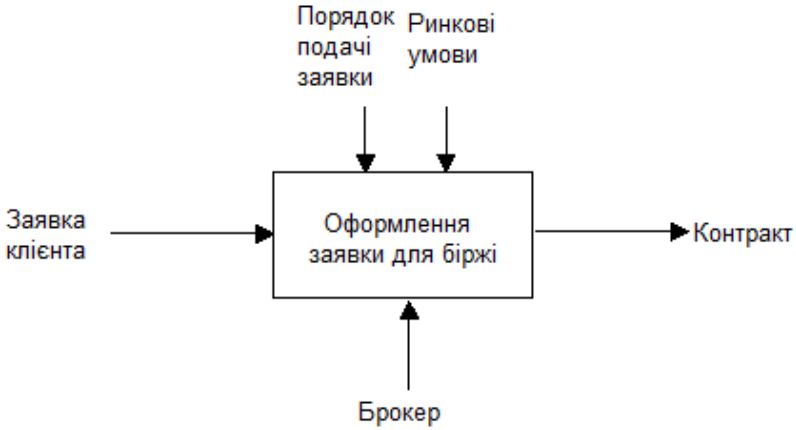

Рисунок 2.2.2.5 - Приклад механізму виконання

Кожен блок на діаграмі має свій номер. Блок будь-якої діаграми може бути далі описаний діаграмою нижнього рівня, яка, в свою чергу, може бути далі деталізована за допомогою необхідного числа діаграм. Таким чином, формується ієрархія діаграм.

Для того, щоб вказати положення довільної діаграми або блоку в ієрархії, використовуються номери діаграм. Наприклад, A21 є діаграмою, яка деталізує блок 1 на діаграмі A2. Аналогічно, A2 деталізує блок 2 на діаграмі A0, яка є самою верхньою діаграмою моделі. На рисунку 2.2.2.6 показано типове дерево ієрархії діаграм.

Рисунок 2.2.2.6 - Ієрархія діаграм

2.2.3. Типи зв’язків між функціями

Одним з важливих моментів при проектуванні ІС за допомогою методології SADT є точна узгодженість типів зв’язків між функціями. Розрізняють сім типів зв’язаності (тип зв’язку/відносна значимість): випадковий/0, логічний/1, часу/2, процедурний/3, комунікаційний/4, послідовний/5, функціональний/6.

Нижче кожен тип зв’язаності коротко визначений і проілюстрований за допомогою типового прикладу з SADT:

- 0) Тип випадкової зв’язаності: найменш бажаний.

Випадкова зв’язаність виникає, коли конкретний зв’язок між функціями незначний або повністю відсутній. Це відноситься до ситуації, коли імена даних на SADT-дугах в одній діаграмі мають незначний зв’язок одне з одним. Крайній варіант цього випадку показаний на рисунку 2.2.3.1.

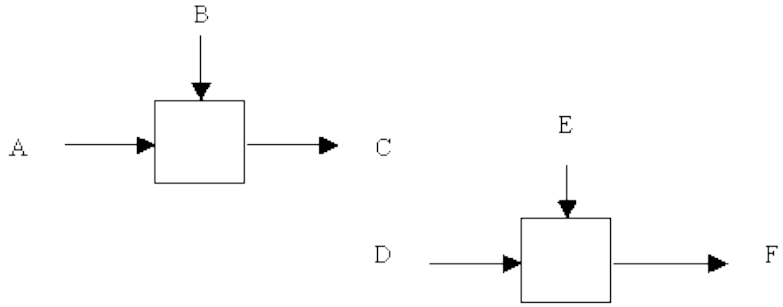

Рисунок 2.2.3.1 - Випадкова зв’язаність

(1) Тип логічної зв’язаності.

Логічне зв’язування відбувається тоді, коли дані і функції збираються разом внаслідок того, що вони потрапляють у загальний клас або набір елементів, але необхідних функціональних відносин між ними не виявляється.

(2) Тип тимчасової зв’язаності.

Пов’язані за часом елементи виникають внаслідок того, що вони представляють функції, пов’язані в часі, коли дані використовуються одночасно або функції включаються паралельно, а не послідовно.

(3) Тип процедурної зв’язаності.

Процедурно пов’язані елементи з’являються згрупованими разом внаслідок того, що вони виконуються протягом однієї і тієї ж частини циклу або процесу. Приклад процедурно зв’язаної діаграми наведений на рисунку 2.2.3.2:

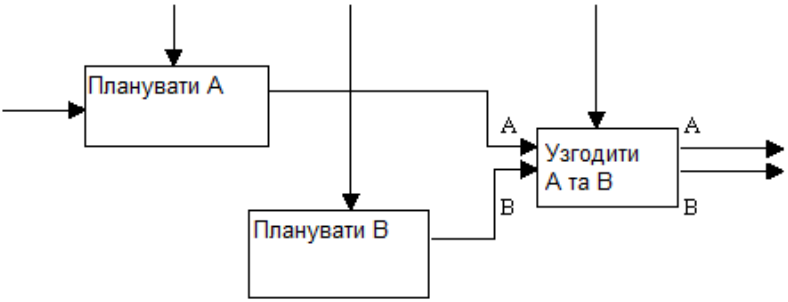

Рисунок 2.2.3.2 - Процедурна зв’язність

(4) Тип комунікаційної зв’язаності.

Діаграми демонструють комунікаційні зв'язки, коли блоки групуються внаслідок того, що вони використовують одні й ті ж вхідні дані та/або виробляють одні й ті ж вихідні дані (рисунок 2.2.3.3).

(5) Тип послідовної зв'язаності.

На діаграмах, що мають послідовні зв'язки, вихід однієї функції служить вхідними даними для наступної функції. Зв'язок між елементами на діаграмі є більш тісним, ніж на розглянутих вище рівнях зв'язку, оскільки моделюються причинно-наслідкові залежності (рисунок 2.2.3.4).

(6) Тип функціональної зв'язності.

Діаграма відображає повну функціональну зв'язаність за наявності повної залежності однієї функції від іншої. Діаграма, яка є чисто функціональною, не містить чужорідних елементів, що відносяться до послідовного або слабшого типу зв'язаності.

Одним із способів визначення функціонально пов'язаних діаграм є розгляд двох блоків, пов'язаних через керуючі дуги, як показано на рисунку 2.2.3.5.

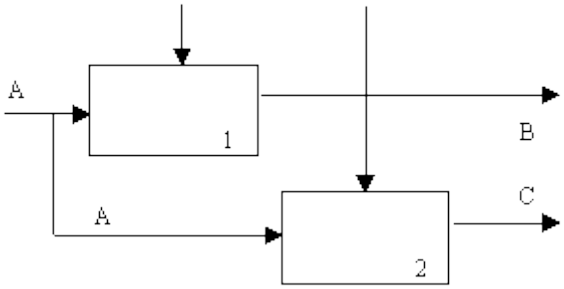

Рисунок 2.2.3.3 - Комунікаційна зв'язаність

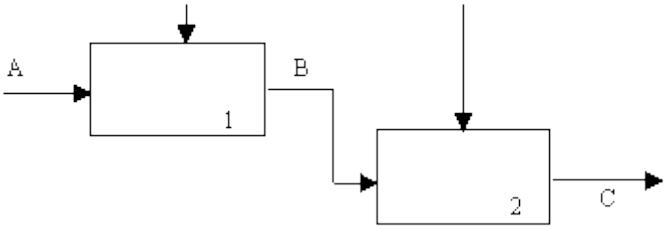

Рисунок 2.2.3.4 - Послідовна зв’язаність

У математичних термінах необхідна умова для найпростішого типу функціональної зв’язаності, показаної на рисунку 2.2.3.5, має наступний вигляд:

$$C = g\,(B) = g\,(f\,(A))$$

Важливо відзначити, що рівні 4-6 встановлюють типи зв’язаності, які розробники вважають найважливішими для одержання діаграм хорошої якості.

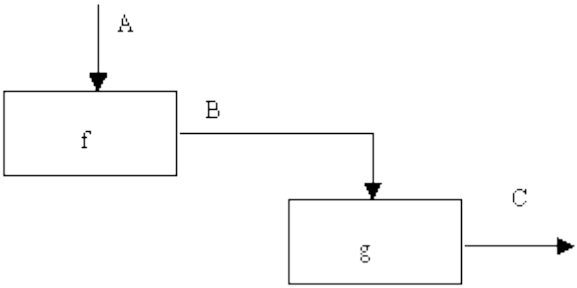

Рисунок 2.2.3.5 - Функціональна зв’язаність

2.3. Моделювання потоків даних (процесів)

В основі даної методології Gane Sarson [6] лежить побудова моделі ІС, яка підлягає аналізу, проектованої або реально існуючої. Відповідно до методології, модель системи визначається, як ієрархія діаграм потоків даних (ДПД або DFD), що описують асинхронний процес перетворення інформації від її введення в систему до видачі користувачу.

Діаграми верхніх рівнів ієрархії (контекстні діаграми) визначають основні процеси або підсистеми ІС з зовнішніми входами і виходами. Вони деталізуються за допомогою діаграм нижнього рівня. Така декомпозиція триває, створюючи багаторівневу ієрархію діаграм, до тих пір, поки не буде досягнуто такий її рівень, на якому процеси стають елементарними і деталізувати їх далі неможливо.

Джерела інформації (зовнішні сутності) породжують інформаційні потоки (потоки даних), які переносять інформацію до підсистем або процесів. Ті, в свою чергу, перетворюють інформацію і породжують нові потоки, які переносять інформацію до інших процесів або підсистем, накопичувачів даних/зовнішніх сутностей, які є споживачами інформації. Таким чином, основними компонентами діаграм потоків даних є:

- зовнішні сутності;
- системи/підсистеми;
- процеси;
- накопичувачі даних;
- потоки даних.

2.3.1. Зовнішні сутності

Зовнішня сутність являє собою матеріальний предмет або фізичну особу, що представляють собою джерело або приймач інформації, наприклад: замовники, персонал, постачальники, клієнти, склад. Визначення деякого об’єкту або системи, як зовнішньої сутності, вказує на те, що вона знаходиться за межами кордонів аналізованої ІС. У процесі аналізу деякі зовнішні сутності можуть бути перенесені всередину діаграми аналізованої ІС, якщо це необхідно, або навпаки: частина процесів ІС може бути винесена за межі діаграми і представлена, як зовнішня сутність.

Зовнішня сутність позначається квадратом (рисунок 2.3.1.1), розташованим немов "над" діаграмою, який кидає на неї тінь, для того, щоб можна було виділити цей символ серед інших позначень:

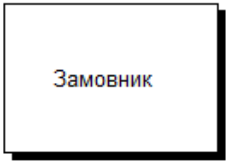

Рисунок 2.3.1.1 - Зовнішня сутність

2.3.2. Системи і підсистеми

При побудові моделі складної ІС вона може бути представлена в самому загальному вигляді на так званій контекстній діаграмі (у вигляді однієї системи як єдиного цілого), або може бути декомпозована на ряд підсистем.

Підсистема (або система) на тематичній діаграмі зображується наступним чином (рисунок 2.3.2.1).

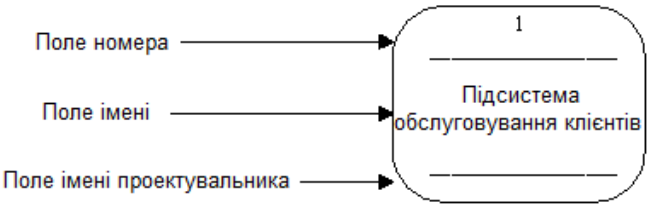

Рисунок 2.3.2.1 - Підсистема

Номер підсистеми слугує для її ідентифікації. У полі імені вводиться найменування підсистеми у вигляді пропозиції з підметом і відповідними визначеннями та доповненнями.

2.3.3. Процеси

Процес є перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму. Фізично процес може бути реалізований різними способами: це може бути підрозділ організації (відділ), що виконує обробку вхідних документів і випуск звітів, програма, апаратно реалізований логічний пристрій тощо.

Процес на діаграмі потоків даних зображується, як показано на рисунку 2.3.3.1:

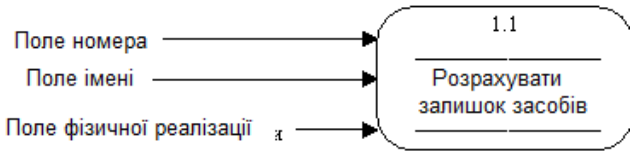

Рисунок 2.3.3.1 - Процес

Номер процесу служить для його ідентифікації. У полі імені вводиться найменування процесу у вигляді пропозиції з активним недвозначним дієсловом у формі (обчислити, розрахувати, перевірити, визначити, створити, отримати), за яким слідують іменники в знахідному відмінку, наприклад:

- "ввести відомості про клієнтів";
- "видати інформацію про поточні витрати";
- "перевірити кредитоспроможність клієнта".

Використання таких дієслів, як "обробити", "модернізувати" або "відредагувати" означає, як правило, недостатньо глибоке розуміння даного процесу і вимагає подальшого аналізу.

Інформація в полі фізичної реалізації показує, який підрозділ організації, програма або апаратний пристрій виконує даний процес.

2.3.4. Накопичувачі даних

Накопичувач даних являє собою абстрактний пристрій для зберігання інформації, який можна в будь-який момент помістити в накопичувач і через деякий час витягнути, причому, способи поміщення та вилучення можуть бути будь-якими.

Накопичувач даних може бути реалізований фізично у вигляді мікроафіші, полиці в картотеці, таблиці в оперативній пам’яті, файлу на магнітному носії тощо. Накопичувач даних на діаграмі потоків даних зображується, як показано на рисунку 2.3.4.1

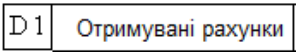

Рисунок 2.3.4.1 - Накопичувач даних

Накопичувач даних ідентифікується буквою "D" і довільним числом. Ім’я накопичувача вибирається з міркування найбільшої інформативності для проектувальника.

Накопичувач даних у загальному випадку є праобразом майбутньої бази даних і опис збережених до неї даних повинен бути пов’язаним з інформаційною моделлю.

2.3.5. Потоки даних

Потік даних визначає інформацію, що була передана через деяке з’єднання від джерела до приймача. Реальний потік даних може бути інформацією, що передається по кабелю між двома пристроями, обміном листами, магнітними або оптичними носіями інформації, які переносяться з одного комп’ютера на інший і т.д.

Потік даних на діаграмі зображується лінією, закінчується стрілкою, що показує напрямок потоку (рисунок 2.3.5.1). Кожен потік даних має ім’я, що відображає його зміст:

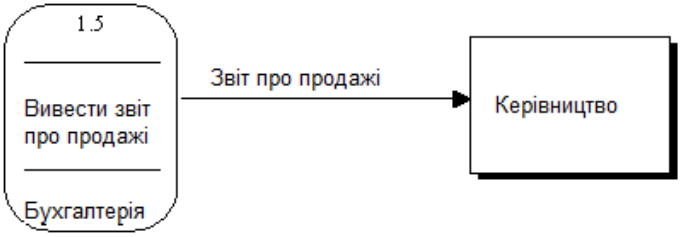

Рисунок 2.3.5.1 - Потік даних

2.3.6. Побудова ієрархії діаграм потоків даних

Першим кроком при побудові ієрархії ДПД є побудова контекстних діаграм. Зазвичай, при проектуванні відносно простих ІС, будується єдина контекстна діаграма із зіркоподібною топологією, в центрі якої знаходиться головний процес, сполучений з приймачами і джерелами інформації, за допомогою яких з системою взаємодіють користувачі та інші зовнішні системи.

Якщо ж для складної системи обмежитися єдиною контекстною діаграмою, то вона буде містити дуже велику кількість джерел і приймачів інформації, які важко розташувати на аркуші паперу нормального формату, і крім того, єдиний головний процес не розкриває структури розподіленої системи. Ознаками складності (в сенсі контексту) можуть бути:

- наявність великої кількості зовнішніх сутностей (десять і більше);

- розподілена природа системи;

- багатофункціональність системи з уже сформованим або виявленим угрупуванням функцій в окремі підсистеми.

Для складних ІС будується ієрархія контекстних діаграм. При цьому, контекстна діаграма верхнього рівня містить не єдиний головний процес, а набір підсистем, з'єднаних потоками даних. Контекстні діаграми наступного рівня деталізують контекст і структуру підсистем.

Ієрархія контекстних діаграм визначає взаємодію основних функціональних підсистем проектованої ІС як між собою, так і з зовнішніми вхідними та вихідними потоками даних та зовнішніми об'єктами (джерелами і приймачами інформації), з якими взаємодіє ІС.

Розробка тематичних діаграм вирішує проблему суворого визначення функціональної структури ІС на самій ранній стадії її проектування, що особливо важливо для складних багатофункціональних систем, у розробці яких беруть участь різні організації та колективи розробників.

Після побудови тематичних діаграм отриману модель слід перевірити на повноту вихідних даних про об'єкти системи та ізольованість об'єктів (відсутність інформаційних зв'язків з іншими об'єктами).

Для кожної підсистеми, присутньої на контекстних діаграмах, виконується її деталізація за допомогою ДПД. Кожен процес на ДПД, в свою чергу, може бути деталізований за допомогою ДПД або мініспецифікації. При деталізації повинні виконуватися наступні правила:

- правило балансування - означає, що при деталізації підсистеми або процесу деталізується діаграма, яка в якості зовнішніх джерел/приймачів даних може мати тільки ті компоненти (підсистеми, процеси, зовнішні сутності, накопичувачі даних), з

якими має інформаційний зв'язок деталізована підсистема або процес на батьківській діаграмі ;

- правило нумерації - означає, що при деталізації процесів повинна підтримуватися їх ієрархічна нумерація. Наприклад, процеси, що деталізують процес з номером 12, отримують номери 12.1, 12.2, 12.3 і т.д.

Мініспецификація (опис логіки процесу) повинна формулювати його основні функції таким чином, щоб надалі фахівець, який виконує реалізацію проекту, зміг виконати їх або розробити відповідну програму.

Мініспецификація є кінцевою вершиною ієрархії ДПД. Рішення про завершення деталізації процесу і використання мініспецификації приймається аналітиком, виходячи з наступних критеріїв:

- наявності у процесі невеликої кількості вхідних і вихідних потоків даних (2-3 потоки);

- можливості опису перетворення процесом даних у вигляді послідовного алгоритму;

- виконання єдиною логічною функцією процесу перетворення вхідної інформації у вихідну;

- можливості опису логіки процесу за допомогою мініспецификації невеликого обсягу (не більше 20-30 рядків).

При побудові ієрархії ДПД переходити до деталізації процесів слід тільки після визначення змісту всіх потоків і накопичувачів даних, які описуються за допомогою структур даних.

Структури даних конструюються з елементів даних і можуть містити:

- альтернативи;

- умовні входження;

- ітерації.

Умовне входження означає, що даний компонент може бути відсутнім у структурі. Альтернатива означає, що в структуру може входити один з перерахованих елементів. Ітерація означає входження будь-якого числа елементів в зазначеному діапазоні.

Для кожного елемента даних може вказуватися його тип (безперервні або дискретні дані). Для безперервних даних може вказуватися одиниця виміру (кг, см тощо), діапазон значень, точність уявлення та форма фізичного кодування. Для дискретних даних може вказуватися таблиця допустимих значень.

Після побудови закінченої моделі системи, її необхідно верифікувати (перевірити на повноту і узгодженість). У повній моделі всі її об’єкти (підсистеми, процеси, потоки даних) повинні бути докладно описані і деталізовані. Виявлені недеталізовані об’єкти слід деталізувати, повернувшись на попередні кроки розробки. У узгодженій моделі для всіх потоків даних і накопичувачів даних має виконуватися правило збереження інформації: всі вхідні куди-небудь дані повинні бути пораховані та систематизовані, а дані, що зчитуються, повинні бути записані.

2.4. Моделювання даних

2.4.1. Case-метод Баркера

Мета моделювання даних полягає в забезпеченні розробника ІС концептуальною схемою бази даних (БД) у формі однієї моделі або кількох локальних моделей, які відносно легко можуть бути відображені в будь-яку систему баз даних.

Найбільш поширеним засобом моделювання даних є діаграми "сутність-зв'язок" (ERD). За їх допомогою визначаються важливі для предметної області об'єкти (сутності), їх властивості (атрибути) і відносини один з одним (зв'язки). ERD безпосередньо використовуються для проектування реляційних баз даних.

Нотація ERD була вперше введена П. Ченом (Chen) і отримала подальший розвиток у роботах Баркера [8]. Метод Баркера буде викладатися на прикладі моделювання діяльності компанії з торгівлі автомобілями. Нижче наведені витяги з інтерв'ю, проведеного з персоналом компанії.

Головний менеджер: один з основних обов'язків - утримання автомобільного майна. Він повинен знати, скільки заплачено за машини і які накладні витрати. Володіючи цією інформацією, він може встановити нижню ціну, за яку міг би продати даний екземпляр. Крім того, він несе відповідальність за продавців і йому потрібно знати, хто що продає і скільки машин продав кожен з них.

Продавець: йому потрібно знати, яку ціну вимагати і яка нижня ціна, за яку можна здійснити операцію. Крім того, йому потрібна основна інформація про машини: рік випуску, марка, модель і т.д.

Адміністратор: його завдання зводиться до складання контрактів, для чого потрібна інформація про покупця, автомашини і продавця, оскільки саме контракти приносять продавцям винагороди за продажі.

Перший крок моделювання - вилучення інформації з інтерв'ю і виділення сутностей.

Сутність (Entity) - реальний або уявний об'єкт, що має істотне значення для даної предметної області, інформація про який підлягає зберіганню (рисунок 2.4.1.1).

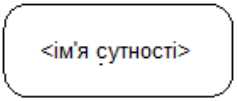

Рисунок 2.4.1.1 - Графічне зображення сутності

Кожна сутність повинна мати унікальний ідентифікатор. Кожен екземпляр сутності повинен однозначно ідентифікуватися і відрізнятися від усіх інших екземплярів даного типу сутності. Кожна сутність повинна володіти деякими властивостями:

- повинна мати унікальне ім'я і до одного і того ж імені повинна завжди застосовуватися одна й та ж інтерпретація. Одна і та ж інтерпретація не може застосовуватися до різних імен, якщо тільки вони не є псевдонімами;

- сутність володіє одним або декількома атрибутами, які або належать сутності, або успадковуються через зв'язок;

- сутність володіє одним або кількома атрибутами, які однозначно ідентифікують кожен її екземпляр;

- кожна сутність може мати будь-яку кількість зв’язків з іншими сутностями моделі.

Звертаючись до наведених вище уривків із інтерв’ю, видно, що сутності, які можуть бути ідентифіковані з головним менеджером - це автомашини і продавці. Продавцю важливі автомашини та пов’язані з їх продажем дані. Для адміністратора важливі покупці, автомашини, продавці і контракти. Виходячи з цього, виділяються 4 сутності (автомашина, продавець, покупець, контракт), які зображуються на діаграмі наступним чином (рисунок 2.4.1.2):

Рисунок 2.4.1.2 - Відображення сутностей

Наступним кроком моделювання є ідентифікація зв’язків.

Зв’язок (Relationship) - поіменована асоціація між двома сутностями, значуща для аналізованої предметної області. Зв’язок - це асоціація між сутностями, при якому, як правило, кожен екземпляр однієї сутності, званої батьківською сутністю, асоційований з довільною (в тому числі - нульовою) кількістю примірників другої сутності, званої сутністю-нащадком, а кожен екземпляр сутності-нащадка асоційований в точності з одним примірником сутності-батька. Таким чином, примірник сутності-нащадка може існувати тільки при існуванні сутності-батька.

Зв’язкам можуть даватися імена, які виражаються граматичним обігом дієслова і розміщаються біля лінії зв’язку. Ім’я

кожного зв’язку між двома даними сутностями повинне бути унікальним, але імена зв’язків у моделі не зобов’язані бути унікальними. Ім’я зв’язку завжди формується з точки зору батьків, так що пропозиція може бути утворена з’єднанням імені сутності-батька, імені зв’язку, виразу ступені та імені сутності-нащадка.

Наприклад, зв’язок продавця і контракту, за яким він працює, може бути виражений таким чином:

- продавець може отримати винагороду за 1 або більше контрактів;

- контракт повинен бути ініційований лише одним продавцем.

Ступінь зв’язку та обов’язковість графічно зображуються наступним чином (рисунок 2.4.1.3):

Рисунок 2.4.1.3 - Ступінь зв’язку та обов’язковість

Таким чином, 2 речення, що описують зв’язок продавця та контракту, графічно будуть виражені наступним чином (рисунок 2.4.1.4):

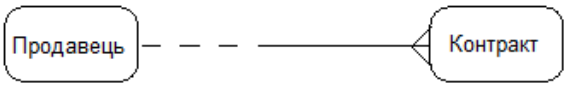

Рисунок 2.4.1.4 - Зображення 2-х речень

Описавши також зв’язки інших сутностей, отримаємо наступну схему (рисунок 2.4.1.5):

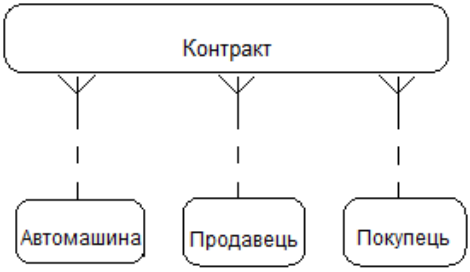

Рисунок 2.4.1.5 - Графічне зображення зв’язків інших сутностей

Останнім кроком моделювання є ідентифікація атрибутів.

Атрибут - будь-яка характеристика сутності, значима для розглянутої предметної області і призначена для кваліфікації, ідентифікації, класифікації, кількісної характеристики або вираження її стану. Атрибут представляє тип характеристик або властивостей, асоційованих з безліччю реальних або абстрактних об’єктів (людей, місць, подій, станів, ідей, пар предметів і т.д.).

Примірник атрибута-це певна характеристика окремого елемента множини. Примірник атрибута визначається типом характеристики та її значенням, званим значенням атрибута. У ER-моделі атрибути асоціюються з конкретними сутностями. Таким чином, примірник сутності повинен володіти єдиним певним значенням для асоційованого атрибута.

Атрибут може бути або обов’язковим, або необов’язковим (рисунок 2.4.1.6). Обов’язковість означає, що атрибут не може приймати невизначених значень (null values). Атрибут може бути або описовим (тобто - звичайним дескриптором сутності), або входити до складу унікального ідентифікатора (первинного ключа).

Унікальний ідентифікатор - це атрибут або сукупність атрибутів і/або зв’язків, призначений для унікальної ідентифікації кожного примірника даного типу сутності. У разі повної ідентифікації кожен примірник даного типу сутності повністю ідентифікується своїми власними ключовими атрибутами, в іншому випадку - в його ідентифікації беруть участь також атрибути іншої сутності-батька (рисунок 2.4.1.7).

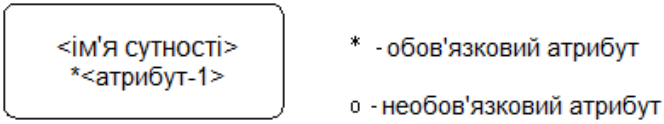

Рисунок 2.4.1.6 - Унікальний ідентифікатор (ідентифікація власними ключовими атрибутами)

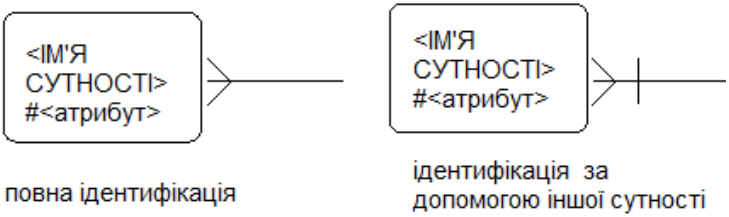

Рисунок 2.4.1.7 - Унікальний ідентифікатор (ідентифікація атрибутами сутності-батька)

Кожен атрибут ідентифікується унікальним ім’ям, висловлюваним граматичним обігом іменника, що описує подану атрибутом характеристику. Атрибути зображуються у списку імен всередині блоку асоційованої суті, причому кожен атрибут займає окремий рядок. Атрибути, що визначають первинний ключ, розміщуються угорі списку і виділяються знаком "#".

Кожна сутність повинна мати хоча б однин можливий ключ. Можливий ключ сутності - це один або декілька атрибутів, чиї значення однозначно визначають кожен екземпляр сутності. При існуванні декількох можливих ключів один з них позначається як первинний ключ, а решта - як альтернативні ключі.

З урахуванням наявної інформації доповнимо побудовану раніше діаграму (рисунок 2.4.1.8).

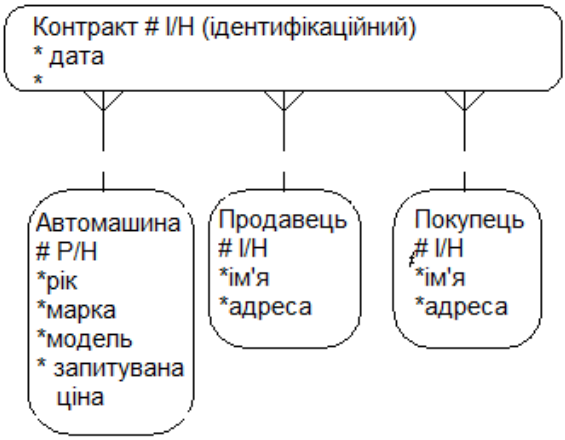

Рисунок 2.4.1.8 - Розширена діаграма процесу

Крім перерахованих основних конструкцій, модель даних може містити ряд додаткових.

- Підтипи і супертипи: одна сутність є узагальнюючим поняттям для групи подібних сутностей (рисунок 2.4.1.9).

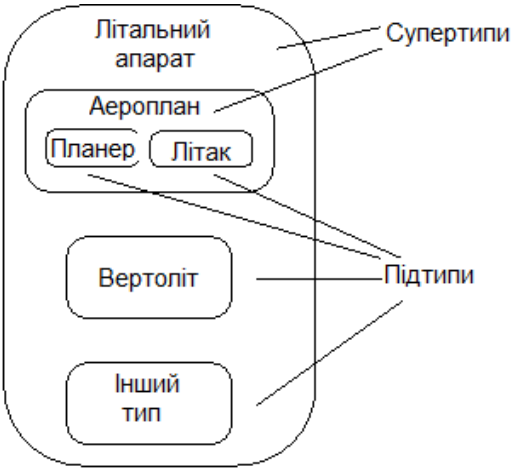

Рисунок 2.4.1.9 - Підтипи і супертипи

- Взаємовиключні зв’язки: кожен екземпляр сутності бере участь тільки в одному зв’язку з групи, взаємно виключаючи інші зв’язки (рисунок 2.4.1.10).

- Рекурсивний зв’язок: сутність може бути пов’язана сама з собою (рисунок 2.4.1.11).

- Непереміщуваний (non-transferrable) зв’язок: примірник сутності не може бути перенесений з одного примірника зв’язку в інший (рисунок 2.4.1.12).

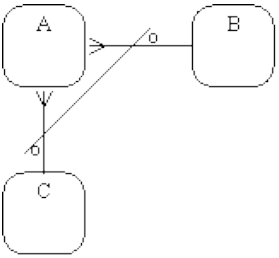

Рисунок 2.4.1.10 - Взаємовиключні зв’язки

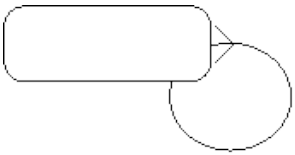

Рисунок 2.4.1.11 - Рекурсивний зв’язок

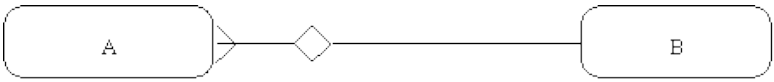

Рисунок 2.4.1.12 - Непереміщуваний зв’язок

2.4.2. Методологія IDEF1

Метод IDEF1, розроблений Т. Ремі (T. Ramey), також заснований на підході П. Чена і дозволяє побудувати модель даних, еквівалентну реляційній моделі в третій нормальній формі.

На сьогодення на основі вдосконалення методології IDEF1 створена її нова версія - методологія IDEF1X. IDEF1X розроблена з урахуванням таких вимог, як простота вивчення і можливість

автоматизації. IDEF1X-діаграми використовуються разом з поширеними CASE-засобами (зокрема, ERwin, Design / IDEF).

Сутність в методології IDEF1X є незалежною від ідентифікаторів або просто незалежною, якщо кожен екземпляр сутності може бути однозначно ідентифікований без визначення його відносин з іншими сутностями. Сутність називається залежною від ідентифікаторів або просто залежною, якщо однозначна ідентифікація примірника суті залежить від його відношення до іншої сутності (рисунок 2.4.2.1).

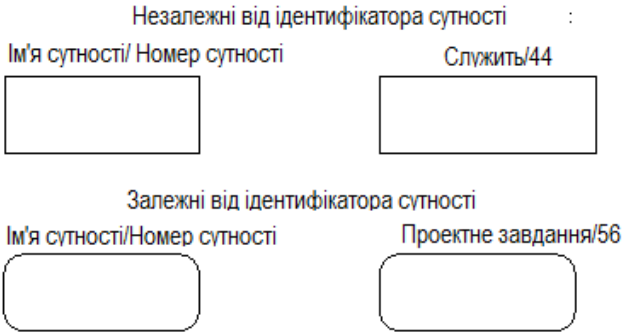

Рисунок 2.4.2.1 - Сутності

Кожній сутності присвоюється унікальне ім’я та номер, які розділяються похилою рискою "/" і розміщаються над блоком.

Зв’язок може додатково визначатися за допомогою вказівки ступені або потужності (кількості примірників сутності-нащадка, яка може існувати для кожного екземпляра сутності-батька). У IDEF1X можуть бути виражені такі потужності зв’язків:

- кожен екземпляр сутності-батька може мати один або більше пов’язаних з ним екземплярів сутності-нащадка або не мати жодного;

- кожен екземпляр сутності-батька повинен мати не менше одного пов’язаного з ним екземпляра сутності-нащадка;

- кожен екземпляр сутності-батька повинен мати не більше одного пов’язаного з ним екземпляра сутності-нащадка;

- кожен екземпляр сутності-батька повинен бути пов’язаний з деяким фіксованим числом примірників сутності-нащадка.

Якщо екземпляр сутності-нащадка однозначно визначається своїм зв’язком з сутністю-батьком, то зв’язок називається ідентифікованим, в іншому випадку - не ідентифікованим і подальшому розгляду в процесі проектування не підлягає.

Зв’язок зображується лінією, проведеною між сутністю-батьком і сутністю-нащадком, з точкою на кінці лінії у сутності-нащадка.

Потужність зв’язку позначається, як показано на рисунку 2.4.2.2 (потужність за замовчуванням - N).

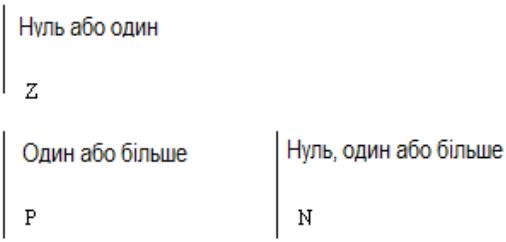

Рисунок 2.4.2.2 - Потужність зв’язку

Ідентифікуючий зв’язок між сутністю-батьком і сутністю-нащадком зображується суцільною лінією (рисунок 2.4.2.3). Сутність-нащадок в ідентифікуючому зв’язку є залежною від ідентифікатора сутністю. Сутність-батько в ідентифікуючому зв’язку може бути як незалежною, так і залежною від ідентифікатора (це визначається її зв’язками з іншими сутностями).

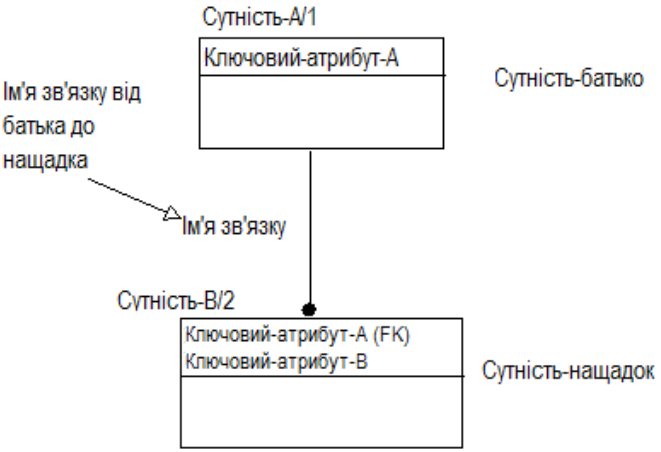

Рисунок 2.4.2.3 - Ідентифікуючий зв’язок

Пунктирна лінія зображує не ідентифікований зв’язок (рисунок 2.4.2.4). Сутність-нащадок в не ідентифікованому зв’язку буде незалежним від ідентифікатора, якщо він не є також сутністю-нащадком в якому-небудь ідентифікуючому зв’язку.

Атрибути зображуються у списку імен всередині блоку сутності. Атрибути, що визначають первинний ключ, розміщуються згори списку, і відділяються від інших атрибутів горизонтальною рискою (рисунок 2.4.2.5):

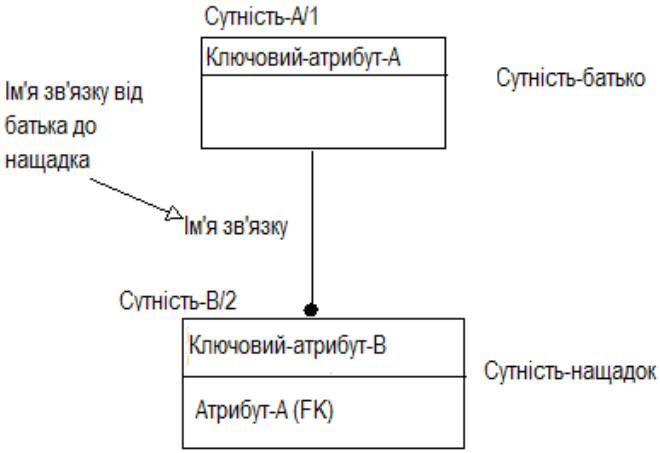

Рисунок 2.4.2.4 - Не ідентифікований зв’язок

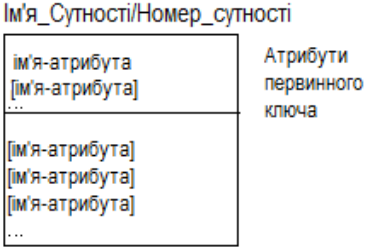

Рисунок 2.4.2.5 - Атрибути і первинні ключі

Сутності можуть мати також зовнішні ключі (Foreign Key), які можуть використовуватися як частини або цілого первинного ключа або неключових атрибутів.

Зовнішній ключ зображується за допомогою приміщення всередині блоку сутності імен атрибутів, після яких ідуть літери FK в дужках (рисунок 2.4.2.6).

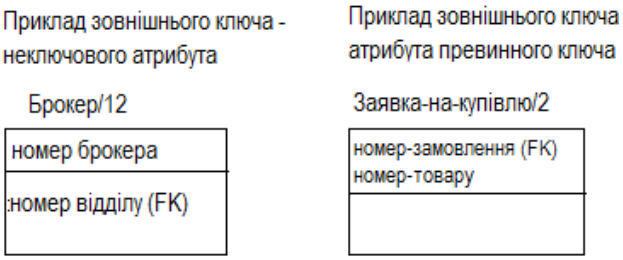

Рисунок 2.4.2.6- Приклади зовнішніх ключів

2.4.3. Підхід, який використовується в CASE-засобі Vantage Team Builder

У CASE-засобі Vantage Team Builder (Westmount I-CASE) [4] використовується один з варіантів нотації П.Чена. На ER-діаграмах сутність позначається прямокутником, що містить ім'я сутності (рисунок 2.4.3.1), а зв'язок - ромбом, пов'язаним лінією з кожною з взаємодіючих сутностей.

Числа над лініями означають ступінь зв'язку.

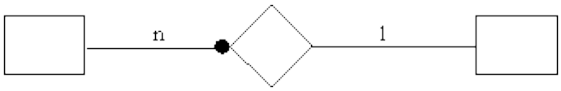

Рисунок 2.4.3.1 - Позначення сутностей і зв'язків

Зв'язки є багатонаправлені і можуть мати атрибути (за винятком ключових). Виділяють два види зв'язків:

- необов'язкова зв'язок (optional);
- слабкий зв'язок (weak).

У необов’язковою зв’язку (рисунок 2.4.3.2) можуть брати участь не всі екземпляри сутності:

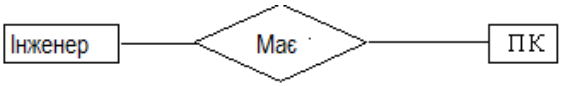

Рисунок 2.4.3.2 - Необов’язковий зв’язок

На відміну від необов’язкового зв’язку, в повному (total) зв’язку беруть участь всі примірники хоча б однієї з сутностей. Це означає, що екземпляри такого зв’язку існують лише за умови існування екземплярів іншої сутності. Повний зв’язок може мати один з 4-х видів:

- обов’язковий зв’язок;
- слабкий зв’язок;
- зв’язок "супертип-підтип";
- асоціативний зв’язок.

Обов’язковий (mandatory) зв’язок описує зв’язок між "незалежною" і "залежною" сутностями. Всі екземпляри залежної ("обов’язкової") сутності можуть існувати тільки за наявності примірників незалежної ("необов’язкової") сутності, тобто - екземпляр "обов’язкової" сутності може існувати тільки за умови існування певного примірника "необов’язкової" сутності.

У прикладі (рисунок 2.4.3.3) мається на увазі, що кожен автомобіль має принаймні одного водія, але не кожен службовець керує машиною.

Рисунок 2.4.3.3 - Обов’язковий зв’язок

У слабкому зв’язку існування однієї з сутностей, що належить деякій множині ("слабкій"), залежить від існування певної сутності, яка належить іншій множині ("сильній"), тобто - екземпляр "слабкої" сутності може бути ідентифікований тільки за допомогою примірника "сильної" сутності. Ключ "сильної" сутності є частиною складного ключа "слабкої" сутності.

Слабкий зв’язок завжди є бінарним і має на увазі обов’язковий зв’язок для "слабкої" сутності. Сутність може бути "слабкою" в одному зв’язку та "сильною" в іншому, але не може бути "слабкою" більше, ніж в одному зв’язку. Слабкий зв’язок може не мати атрибутів.

Приклад на рисунку 2.4.3.4: ключ (номер) рядка в документі може не бути унікальним і повинен бути доповнений ключем документа.

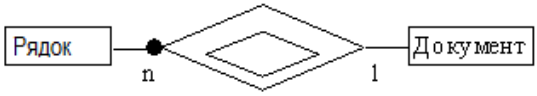

Рисунок 2.4.3.4 - Слабкий зв’язок

Зв’язок "супертип-підтип" зображено на рисунку 2.4.3.5. Загальні характеристики (атрибути) типу визначаються по сутності-супертипу, сутність-підтип успадковує всі характеристики супертипу.

Примірник підтипу існує тільки за умови існування певного примірника супертипу. Підтип не може мати ключа (він імпортує ключ з супертипу). Сутність, яка є супертипом в одному зв’язку, може бути підтипом в іншому зв’язку. Зв’язок супертипу не може мати атрибутів.

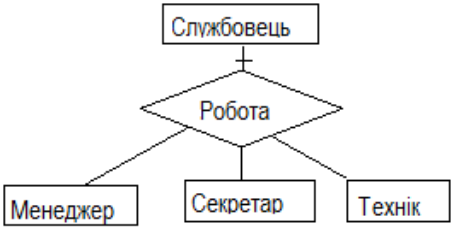

Рисунок 2.4.3.5 - Зв’язок "супертип-підтип"

У асоціативному зв’язку кожен екземпляр зв’язку (асоціативний об’єкт) може існувати тільки за умови існування певних примірників кожної з взаємопов’язаних сутностей. Асоціативний об’єкт - об’єкт, що є одночасно сутністю і зв’язком. Асоціативний зв’язок - це зв’язок між кількома "незалежними" сутностями і однією "залежною" сутністю. Зв’язок між незалежними сутностями має атрибути, які визначаються в залежній сутності. Таким чином, залежна сутність визначається в термінах атрибутів зв’язку між іншими сутностями.

У прикладі на рисунку 2.4.3.6 літак виконує посадку на злітну смугу в заданий час при певній швидкості і напрямку вітру. Оскільки ці характеристики застосовні тільки до конкретної посадки, вони є атрибутами посадки, а не літака або злітної смуги.

Пілот, що виконує посадку, пов’язаний набагато сильніше з конкретною посадкою, ніж з літаком або злітною смугою.

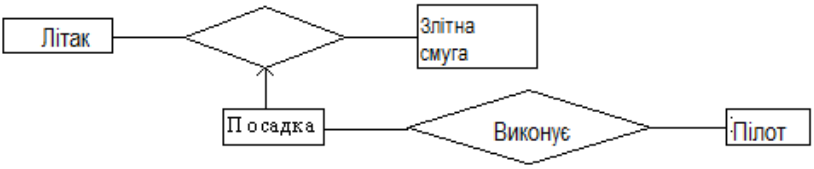

Рисунок 2.4.3.6 - Асоціативний зв’язок

Первинний ключ кожного типу сутності позначається зірочкою (*).

ER-діаграма має підпорядковуватися наступним правилам:

- кожна сутність, кожен атрибут і кожен зв’язок повинні мати ім’я (зв’язок супертипу або асоціативний зв’язок може не мати імені);

- ім’я сутності повинне бути унікальним в рамках моделі даних;

- ім’я атрибута повинне бути унікальним в рамках сутності;

- ім’я зв’язку повинне бути унікальним, якщо для нього генерується таблиця БД;

- кожен атрибут повинен мати визначення типу даних;

- сутність в необов’язковому зв’язку повинна мати ключовий атрибут.

Те ж саме відноситься до:

- сильної сутності в слабкому зв’язку;

- супертипу у зв’язку "супертип-підтип";

- необов’язкової сутності в обов’язковому (повному) зв’язку.

Підтип у зв’язку "супертип-підтип" не може мати ключовий атрибут.

В асоціативному або слабкому зв’язку може бути тільки одна асоціативна (слабка) сутність. Зв’язок не може бути одночасно обов’язковим, "супертипом-підтипом" або асоціативним.

2.5. Приклад використання структурного підходу

2.5.1. Опис предметної області

У даному прикладі використовується методологія Yourdon [2], реалізована в CASE-засобі Vantage Team Builder [4].

В якості предметної області використовується опис роботи відеобібліотеки, яка отримує запити на фільми від клієнтів і носії інформації (НСІ), що повертаються клієнтами.

Запити розглядаються адміністрацією відеобібліотеки з використанням інформації про клієнтів, фільми та стрічки. При цьому перевіряється і оновлюється список орендованих НСІ, а також перевіряються записи про членство в бібліотеці. Адміністрація контролює також повернення НСІ, використовуючи інформацію про фільми, НСІ і список орендованих НСІ, що відновлюється постійно в динамічному режимі.

Обробка запитів на фільми і повернення НСІ включає наступні дії: якщо клієнт не є членом бібліотеки, він не має права на оренду. Якщо потрібний фільм є в наявності, адміністрація інформує клієнта про орендну плату. Однак, якщо клієнт прострочив термін повернення наявних у нього НСІ, йому не дозволяється брати нові фільми. Коли носії інформації

повертаються, адміністрація розраховує орендну плату плюс пеню за несвоєчасне повернення.

Відеобібліотека отримує нові носії інформації від своїх постачальників. Коли нові носії інформації надходять до бібліотеки, необхідна інформація про них фіксується. Інформація про членство в бібліотеці міститься окремо від записів про оренду носіїв інформації.

Адміністрація бібліотеки регулярно готує звіти за певний період часу про членів бібліотеки, постачальників носіїв інформації, видачі певних носіїв інформації і про носії інформації, придбанні бібліотекою.

2.5.2. Організація проекту

Весь проект розділяється на 4 фази:

- аналіз;
- глобальне проектування (проектування архітектури системи);
- детальне проектування;
- реалізація (програмування).

На фазі аналізу будується модель середовища (Environmental Model). Побудова моделі середовища включає:

- аналіз поведінки системи (визначення призначення ІС, побудова початкової контекстної діаграми потоків даних (DFD) та формування матриці списку подій (ELM), побудову тематичних діаграм);
- аналіз даних (визначення складу потоків даних та побудова діаграм структур даних (DSD),

- конструювання глобальної моделі даних у вигляді ER-діаграми.

Призначення ІС визначає наявність між проектувальниками і замовниками узгодження щодо призначення майбутньої ІС, загальний опис ІС для самих проектувальників і її межі. Призначення фіксується як текстовий коментар в "нульовому" процесі контекстної діаграми.

Наприклад, в даному випадку призначення ІС формулюється наступним чином:

- ведення бази даних щодо членів бібліотеки;
- ведення бази даних щодо наявних фільмів;
- ведення бази даних щодо оренди;
- ведення бази даних щодо постачальників.

При цьому, керівництво бібліотеки повинне мати можливість отримувати різні види звітів для виконання своїх завдань.

Перед побудовою контекстної DFD необхідно проаналізувати зовнішні події (зовнішні об’єкти), які мають вплив на функціонування бібліотеки. Ці об’єкти взаємодіють з ІС шляхом інформаційного обміну з нею.

З опису предметної області випливає, що в процесі роботи бібліотеки беруть участь такі групи людей:

- клієнти;
- постачальники;
- керівництво.

Ці групи є зовнішніми об’єктами. Вони не тільки взаємодіють з системою, але і визначають її межі та зображуються на початковій контекстній DFD в якості термінаторів (зовнішні сутності).

Початкова контекстна діаграма зображена на рисунку 2.5.2.1. На відміну від нотації Gane/Sarson, зовнішні сутності позначаються звичайними прямокутниками, а процеси - колами.

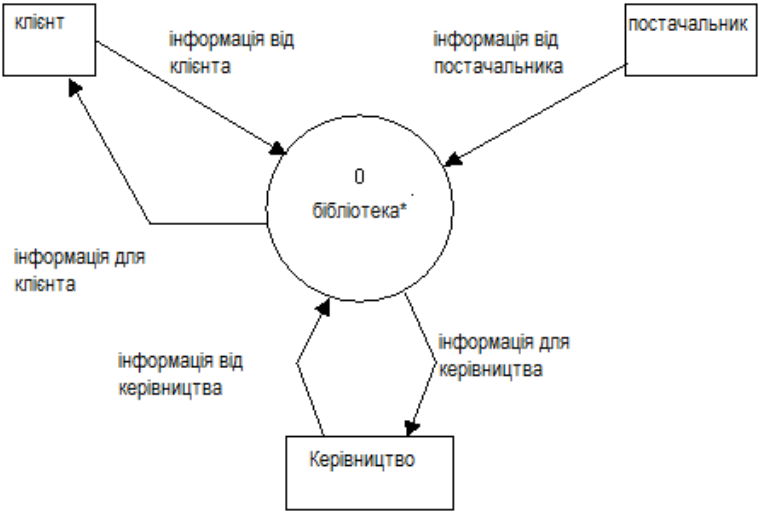

Рисунок 2.5.2.1 - Початкова контекстна діаграма

Список подій будується у вигляді матриці (ELM) і описує різні дії зовнішніх сутностей та реакцію ІС на них. Ці дії являють собою зовнішні події, що впливають на роботу бібліотеки. Розрізняють такі типи подій:

- NC - нормальне керування;
- ND - нормальні дані;
- NCD - нормальне керування/дані;
- TC - тимчасове керування;
- TD - тимчасові дані;
- TCD - тимчасове керування/дані.

Всі дії позначаються як нормальні дані. Ці дані є подіями, які ІС сприймає безпосередньо, наприклад, зміна адреси клієнта, яка повинна бути відразу зареєстрована. Вони з'являються в DFD в якості вмісту потоків даних.

Матриця списку подій має наступний вигляд:

- клієнт бажає стати членом бібліотеки (ND - реєстрація клієнта як члена бібліотеки);

- клієнт повідомляє про зміну адреси (ND - реєстрація зміненої адреси клієнта);

- клієнт надає запит про оренду фільму (ND - розгляд запиту);

- клієнт повертає фільм (ND - реєстрація повернення);

- керівництво надає повноваження новому постачальнику (ND - реєстрація постачальника);

- постачальник повідомляє про зміну адреси (ND - реєстрація зміненої адреси постачальника);

- постачальник направляє фільм в бібліотеку (ND - отримання нового фільму);

- керівництво запитує новий звіт (ND - формування необхідного звіту для керівництва).

Для завершення аналізу функціонального аспекту поведінки системи будується повна контекстна діаграма, що включає діаграму нульового рівня. При цьому, процес "бібліотека" декомпозується на 4-и процеси, що відображають основні види адміністративної діяльності бібліотеки. Існуючі "абстрактні" потоки даних між термінаторами і процесами трансформуються в потоки, що представляють обмін даними на більш конкретному рівні. Список подій показує, які потоки існують на цьому рівні, кожна подія зі

списку має формувати деякий потік (подія формує вхідний потік, реакція - вихідний потік). Один "абстрактний" потік може бути поділений на більш ніж один "конкретний" потік.

На наведеній DFD (рисунок 2.5.2.2) накопичувач даних "бібліотека" є глобальним або абстрактним поданням сховища даних.

Аналіз функціонального аспекту поведінки системи дає уявлення про обмін та перетворення даних в системі. Взаємозв’язок між "абстрактними" потоками даних та "конкретними" потоками даних на діаграмі нульового рівня виражається в діаграмах структур даних (рисунок 2.5.2.3).

На фазі аналізу будується глобальна модель даних, яка надається у вигляді діаграми "сутність-зв’язок" (рисунок 2.5.2.4).

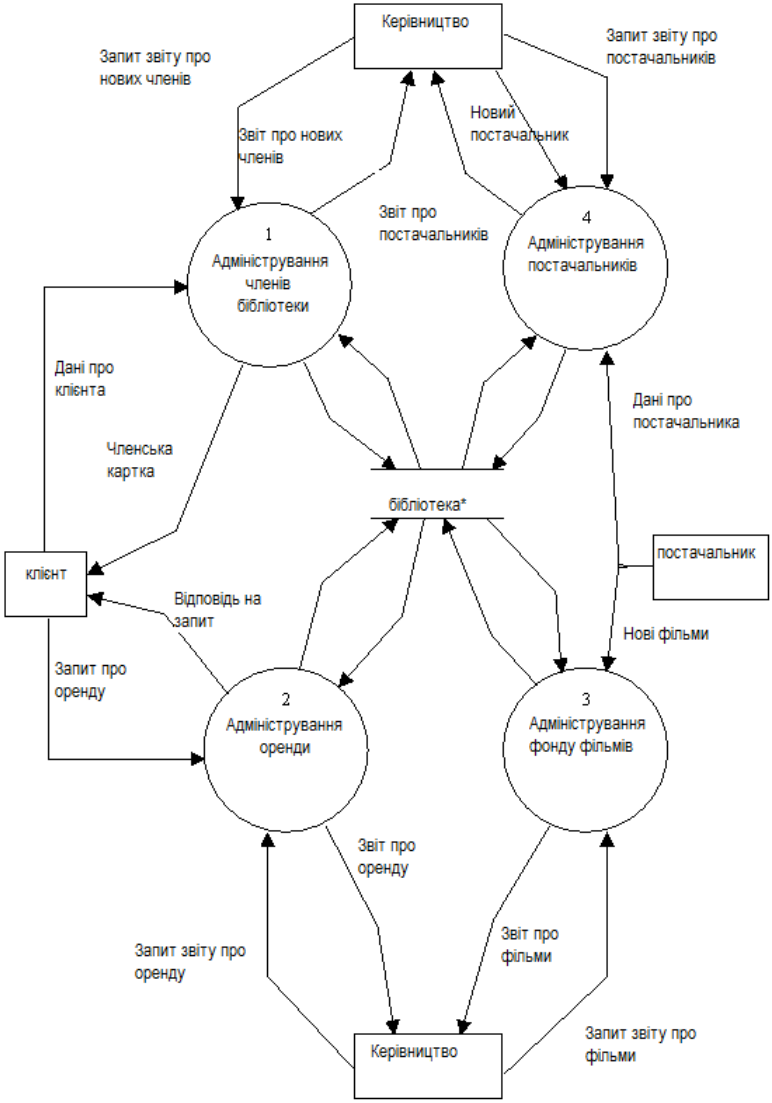

Рисунок 2.5.2.2 - Контекстна діаграма

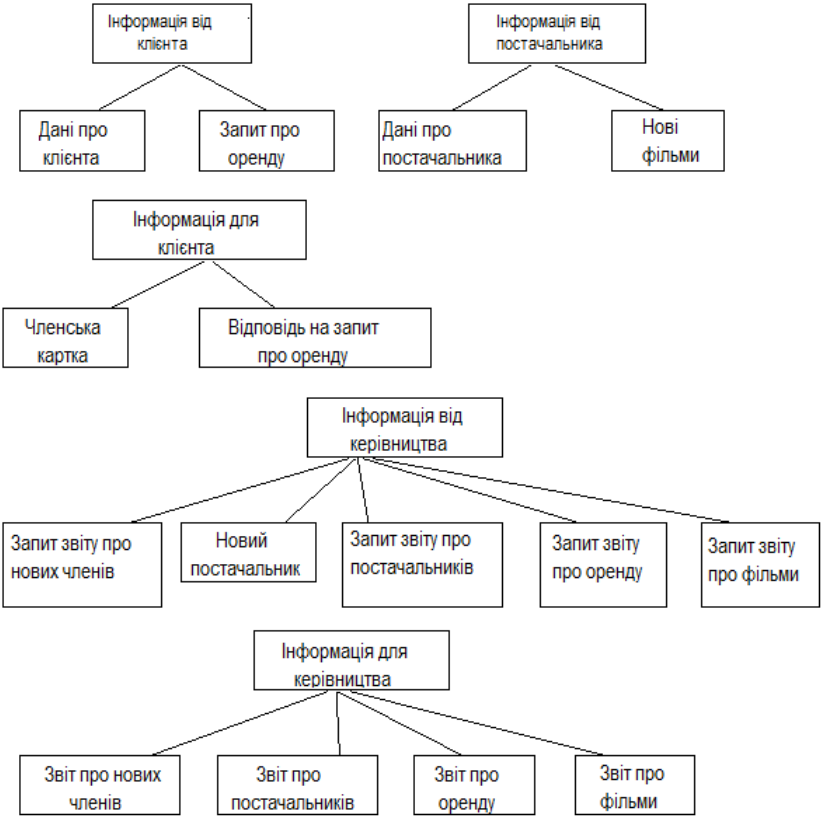

Рисунок 2.5.2.3 - Діаграма структур даних

Між різними типами діаграм існують наступні взаємозв'язки:

- ELM-DFD: події - вхідні потоки, реакції - вихідні потоки;

- DFD-DSD: потоки даних - структури даних верхнього рівня;

- DFD-ERD: накопичувачі даних - ER-діаграми;

- DSD-ERD: структури даних нижнього рівня - атрибути сутностей.

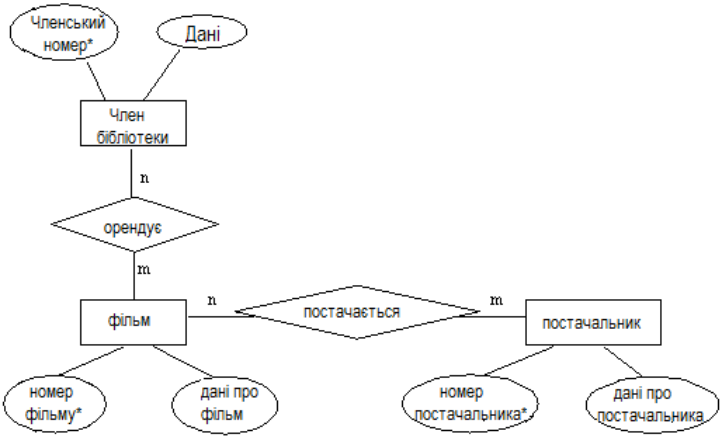

Рисунок 2.5.2.4- Діаграма "сутність-зв'язок"

На фазі проектування архітектури будується предметна модель. Процес побудови предметної моделі включає в себе:

- детальний опис функціонування системи;

- подальший аналіз даних, які використовуються;

- побудова логічної моделі даних для подальшого проектування бази даних (БД);

- визначення структури для користувача інтерфейсу;

- специфікації форм та упорядкування їх появи;

- уточнення діаграм потоків даних і списку подій;

- виділення серед процесів нижнього рівня інтерактивних і не інтерактивних;

- визначення для них мініспецифікації.

Результатами проектування архітектури є:

- модель процесів (діаграми архітектури системи (SAD) і мініспецифікації на структурованій мові);

- модель даних (ERD) і підсхеми (ERD);

- модель користувацького інтерфейсу (класифікація процесів на інтерактивні та не інтерактивні функції, діаграма послідовності форм (FSD - Form Sequence Diagram), що показує, які форми з'являються в додатку і в якому порядку. На FSD фіксується набір і структура викликів екранних форм. Діаграми FSD утворюють ієрархію, на вершині якої знаходиться головна форма програми, що реалізує підсистему. На другому рівні знаходяться форми, що реалізують процеси нижнього рівня функціональної структури, зафіксованої на діаграмах SAD.

На фазі детального проектування будується модульна модель. Під модульною моделлю розуміється реальна модель проектованої прикладної системи. Процес її побудови включає в себе:

- уточнення моделі бази даних для подальшої генерації SQL-пропозицій;

- уточнення структури для користувача інтерфейсу;

- побудова структурних схем, що відображають логіку роботи призначеного для користувача інтерфейсу, і моделі бізнес-логіки (Structure Charts Diagram - SCD) та прив'язка їх до форм.

Результатами детального проектування є:

- модель процесів (структурні схеми інтерактивних і не інтерактивних функцій);

- модель даних (визначення в ERD всіх необхідних параметрів для додатків);

- модель користувацького інтерфейсу (діаграма послідовності форм (FSD), що показує, які форми з'являються в додатку і в якому порядку, взаємозв'язок між кожною формою і певною структурною схемою, взаємозв'язок між кожною формою і однією або більше сутностями в ERD).

На фазі реалізації будується реалізаційна модель. Процес її побудови включає в себе:

- генерацію SQL-пропозицій, що визначають структуру цільової БД (таблиці, індекси, обмеження цілісності);

- уточнення структурних схем (SCD) і діаграм послідовності форм (FSD) з наступною генерацією коду додатків.

На основі аналізу потоків даних і взаємодії процесів зі сховищами даних, здійснюється остаточне виділення підсистем (попереднє повинне було бути зроблене і зафіксоване на етапі формулювання вимог в технічному завданні).

При виділенні підсистем необхідно керуватися принципом функціональної зв'язаності і принципом мінімізації інформаційної залежності. Необхідно враховувати, що на підставі таких елементів підсистеми, як процеси і дані, на етапі розробки повинна бути створена програма, здатна функціонувати самостійно.

З іншого боку, при угрупуванні процесів і даних в підсистеми необхідно враховувати вимоги до конфігурації продукту, якщо вони були сформульовані на етапі аналізу.

3. Програмні засоби підтримки життєвого циклу ПЗ

3.1. Методології проектування ПЗ як програмні продукти. Методологія DATARUN та інструментальний засіб SE Companion

Сучасні методології та технології, які їх реалізують, поставляються в електронному вигляді разом з CASE-засобами і включають бібліотеки:

- процесів;
- шаблонів;
- методів;
- моделей та інших компонентів, призначених для побудови ПЗ такого класу систем, на який орієнтована методологія. Електронні методології включають також кошти, які повинні забезпечувати їх адаптацію для конкретних користувачів і розвиток методології за результатами виконання конкретних проектів.

Процес адаптації полягає в наступному:

- у видаленні непотрібних процесів, дій ЖЦ та інших компонентів методології;
- у зміні невідповідних процесів, дій ЖЦ та інших компонентів методології;
- в додаванні власних процесів і дій;
- в додаванні методів, моделей, стандартів та настанов.

Налаштування методології може здійснюватися також за такими аспектами:

- етапи та операції ЖЦ;
- учасники проекту;

- використовувані моделі ЖЦ;

- підтримувані концепції та ін.

Електронні методології та технології (і підтримуючі їх CASE-засоби) складають ядро комплексу узгоджених інструментальних засобів середовища розробки ІС.

3.1.1. Методологія DATARUN

Однією з найбільш поширених в світі електронних методологій є методологія DATARUN [6]. Відповідно до методології DATARUN, ЖЦ ПЗ розбивається на стадії, які зв'язуються з результатами виконання основних процесів, визначених стандартом ISO 12207. Кожна стадія, окрім отримання власних результатів, має завершувати план роботи на наступну стадію.

Стадія формування вимог і планування включає в себе дії з визначенням початкових оцінок обсягу і вартості проекту. Повинні бути сформульовані:

- вимоги до проекту, який підлягає розробці;

- економічне обгрунтування проекту;

- функціональні моделі майбутньої ІС (моделі бізнес-процесів організації);

- вихідна концептуальна модель даних, які дають основу для оцінки технічної реалізованості проекту.

Основними результатами цієї стадії повинні бути:

- моделі діяльності організації (вихідні моделі процесів і даних організації);

- вимоги до системи, включаючи вимоги по спряженню з існуючими ІС;

- початковий бізнес-план.

Стадія концептуального проектування починається з детального аналізу первинних даних та уточнення концептуальної моделі даних, після чого проектується архітектура системи.

Архітектура включає в себе поділ концептуальної моделі на доступні для огляду підмоделі. Оцінюється можливість використання існуючих ІС та вибирається відповідний метод їх перетворення. Після побудови проекту уточнюється вихідний бізнес-план. Вихідними компонентами цієї стадії є:

- концептуальна модель даних;

- модель архітектури системи;

- уточнений бізнес-план.

На стадії специфікації додатків триває процес створення і деталізації проекту. Концептуальна модель даних перетворюється в реляційну модель даних. Визначається структура програми, необхідні інтерфейси програми у вигляді екранів, звітів та пакетних процесів разом з логікою їх виклику.

Модель даних уточнюється бізнес-правилами і методами для кожної таблиці. У кінці цієї стадії приймається остаточне рішення про спосіб реалізації програм. За результатами стадії повинен бути побудований проект ІС, до складу якого входять моделі:

- архітектури ІС;

- даних;

- функцій;

- інтерфейсів (із зовнішніми системами, із користувачами);

- вимог до розроблюваних програм (моделі даних, інтерфейсів і функцій);

- вимог з доопрацювання існуючих ІС;

- вимог до інтеграції додатків;

- сформований остаточний план створення ІС.

На стадії розробки, інтеграції і тестування повинна бути створена тестова база даних, приватні та комплексні тести. Проводиться розробка, прототипування і тестування баз даних та програм відповідно до проекту. Налагоджується робота інтерфейсів з існуючими у замовника системами. Описується конфігурація поточної версії ПЗ. На основі результатів тестування проводиться оптимізація бази даних та програм. Програми інтегруються в систему, проводиться тестування додатків в складі системи та випробування системи. Основними результатами стадії є:

- готові програми, перевірені в складі системи на комплексних тестах;

- поточний опис конфігурації ПЗ;

- скорегована за результатами випробувань версія системи;

- експлуатаційна документація на систему.

Стадія впровадження включає в себе дії зі встановлення та впровадження баз даних і додатків. Основними результатами стадії повинні бути:

- готова до експлуатації і перенесена на програмно-апаратну платформу замовника версія системи;

- документація супроводження системи в промисловій експлуатації;

- акт приймальних випробувань за результатами дослідної експлуатації.

Стадії супроводження та подальшого розвитку системи в процесі ЖЦ включають:

- процеси та операції, пов’язані з реєстрацією, діагностикою і локалізацією помилок та збоїв в роботі системи впродовж її ЖЦ;

- внесення відповідних змін у вигляді введення нових компонентів системи, розширення/покращення її функціональних можливостей, в разі вимог та пропозицій замовника програмно-апаратного комплексу, тощо;

- тиражування та розповсюдження нових версій ПЗ в місця його експлуатації,

- переміщення додатків на нову платформу;

- масштабуванням системи.

Стадія розвитку фактично є повторною ітерацією стадії розробки.

Методологія DATARUN спирається на дві моделі або на два подання:

- модель організації;

- модель ІС.

Методологія DATARUN базується на системному підході до опису діяльності організації. Побудова моделей починається з опису процесів, з яких потім витягуються первинні дані (стабільна підмножина даних, які організація повинна використовувати для своєї діяльності).

Первинні дані описують ОП або послуги організації, що виконують операції (транзакції) і споживані ресурси. До первинних відносяться дані, які характеризують зовнішні і внутрішні сутності, такі як службовці, клієнти або агентства, а також дані, отримані в

результаті прийняття рішень, як наприклад, графіки робіт, ціни на продукти.

Основний принцип DATARUN полягає в тому, що первинні дані, якщо вони належним чином організовані в модель даних, стають основою для проектування архітектури ІС. Архітектура ІС буде більш стабільною, якщо вона заснована на первинних даних, тісно пов'язаних з основними діловими операціями, що визначаються природою архітектури сфери діяльності замовника, а не традиційними функціональними моделями.

Будь-яка ІС (рисунок 3.1.1.1) являє собою набір модулів, що виконуються процесорами при взаємодії з базами даних. Бази даних і процесори можуть розташовуватися централізовано або бути розподіленими. Події в системі можуть ініціюватися зовнішніми сутностями, такими як клієнти біля банкоматів або тимчасові події (кінець місяця чи кварталу). Усі транзакції здійснюються через об'єкти або модулі інтерфейсу, які взаємодіють з однією або більше базами даних.

Підхід DATARUN переслідує дві мети:

- визначити стабільну структуру, на основі якої буде будуватися ІС. Такою структурою є модель даних, отримана з первинних даних, яка представляє собою фундаментальні процеси організації;

- спроектувати ІС на основі моделі даних.

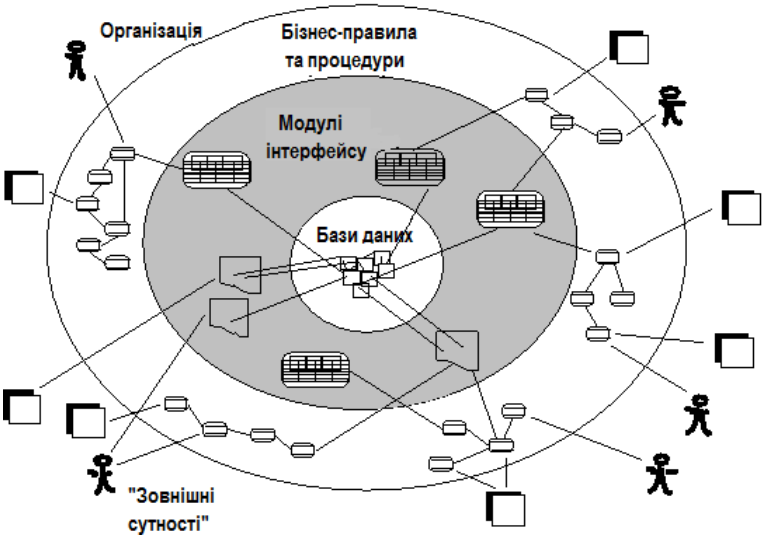

Рисунок 3.1.1.1 - Модель ІС

Об’єкти, що формуються на основі моделей даних, є об’єктами бази даних, зазвичай розміщеними на серверах в середовищі клієнт/сервер. Об’єкти інтерфейсу, визначені в архітектурі комп’ютерної системи (КС), зазвичай розміщуються на клієнтській частині. Модель даних, що є основою для специфікації спільно використовуваних об’єктів бази даних і різних об’єктів інтерфейсу, забезпечує супроводження ІС. На рисунку 3.1.1.2 представлена послідовність кроків проектування ІС.

На малюнку 3.1.1.3 визначені моделі, які створюються в процесі розробки ІС. Для їх створення використано CASE-засіб Silverrun, описаний в підрозділі 5.1. Silverrun забезпечує автоматизацію проведення проектних робіт відповідно до

методології DATARUN. Надане цими засобами середовище проектування дає можливість керівнику проекту

- контролювати проведення робіт;

- відстежувати виконання робіт;

- вчасно помічати відхилення від графіка.

Кожен учасник проекту, підключившись до цього середовища, може:

- з'ясувати зміст і терміни виконання дорученої йому роботи;

- детально вивчити техніку її виконання в гіпертексті за технологіями;

- викликати інструмент (модуль Silverrun) для реального виконання роботи.

Інформаційна система, що створюється послідовною побудовою ряду моделей, починаючи з моделі бізнес-процесів і закінчуючи моделлю програми, автоматизує ці процеси, зводячи до мінімуму в процесі розробки людський фактор. Це досягається шляхом використання готових модулів, які надає методологія DATARUN:

- BPM (Business Process Model) - модель бізнес-процесів;.

- PDS (Primary Data Structure) - структура первинних даних;

- CDM (Conceptual Data Model) - концептуальна модель даних;

- SPM (System Process Model) - модель процесів системи;

- ISA (Information System Architecture) - архітектура інформаційної системи;

- ADM (Application Data Model) - модель даних програми;

Рисунок 3.1.1.2 - Послідовність кроків проектування системи

- IPM (Interface Presentation Model) - модель представлення інтерфейсу;

- ISM (Interface Specification Model) - модель специфікації інтерфейсу.

Створювана ІС повинна ґрунтуватися на функціях, що виконуються організацією. Тому перша створювана модель - це модель бізнес-процесів, побудова якої здійснюється в модулі Silverrun BPM. Для цієї моделі використовується спеціальна нотація BPM. У процесі аналізу і специфікації бізнес-функцій виявляються основні інформаційні об’єкти, які документуються як структури даних, пов’язані з потоками і сховищами моделі.

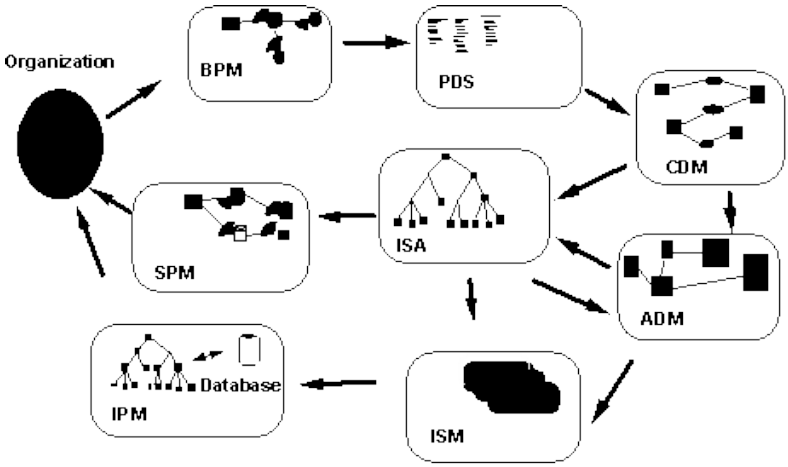

Рисунок 3.1.1.3 - Моделі, що створюються за допомогою підходу DATARUN

Джерелами для створення структур є:

- документи (які використовуються в організації);
- посадові інструкції;
- описи виробничих операцій.

Ці дані вводяться в тому вигляді, у якому вони існують в діяльності організації. Нормалізація і видалення надлишкових даних проводиться пізніше при побудові концептуальної моделі даних в модулі Silverrun ERX. Після створення моделі бізнес-процесів інформація зберігається в репозиторії проекту.

У процесі обстеження роботи організації виявляються і документуються структури первинних даних. Ці структури заносяться до головного сховища модуля BPM і при описі циркулюють в організації документів, повідомлень, даних. У

моделі бізнес-процесів первинні структури даних пов'язані з потоками і сховищами інформації.

На основі структур первинних даних в модулі Silverrun ERX створюється концептуальна модель даних (ER-модель). Від структур первинних даних концептуальна модель відрізняється видаленням надлишкових даних, стандартизацією найменувань понять і нормалізацією. Ці операції в модулі ERX виконуються за допомогою вбудованої експертної системи. Мета концептуальної моделі даних - описати інформацію, яка використовується, без деталей можливої реалізації в базі даних але в добре структурованому нормалізованому вигляді.

На основі моделі бізнес-процесів і концептуальної моделі даних проектується архітектура ІС. Для кожної програми специфікуються дані, що використовуються і реалізуються функції. Архітектура ІС створюється в модулі Silverrun BPM з використанням спеціальної нотації ISA. Основний зміст цієї моделі - структурні компоненти системи і навігація між ними. Концептуальна модель даних розбивається на частини, які відповідно входять до складу системи додатків.

Перед розробкою додатків повинна бути спроектована структура корпоративної бази даних. DATARUN передбачає використання бази даних, розробленої на основі реляційній моделі.

RDM. Перетворення моделі з формату ERX в формат RDM відбувається автоматично без втручання користувача. Після перетворення форматів він одержить модель реляційної бази даних. Ця модель деталізується в модулі Silverrun RDM визначенням фізичної реалізації (типів даних СУБД, ключів, індексів, тригерів, різноманітних обмежень, тощо).

Правила обробки даних можна задавати як безпосередньо на мові програмування СУБД, так і в декларативній формі, не прив'язаній до реалізації. Мости Silverrun до реляційних СУБД перекладають ці декларативні правила на мову системи/ОП, які знаходяться в розробці. Це дозволяє знизити трудомісткість програмування процедур бази даних сервера, а також дозволяє з однієї специфікації генерувати програми для різних СУБД.

За допомогою моделі системних процесів детально документується поведінка кожної програми. У модулі BPM створюється модель системних процесів, яка визначає, яким чином реалізуються бізнес-процеси. Ця модель створюється окремо для кожного додатка і тісно пов'язана з моделлю даних програми.

Додаток складається з інтерфейсних об'єктів:

- екранних форм;

- звітів;

- процедур обробки даних.

Кожен інтерфейс системи (екранна форма, звіт, процедура обробки даних) має справу з підмножиною бази даних. У моделі даних програми (створеної в модулі RDM) створюється підсхема бази даних для кожного інтерфейсу цієї програми. Уточнюються також правила обробки даних, специфічні для кожного інтерфейсу. Інтерфейс працює з даними в ненормалізованому вигляді, тому специфікація даних (якою її бачить інтерфейс) оформляється як окрема підсхема моделі даних інтерфейсу.

Модель представлення інтерфейсу - це опис зовнішнього вигляду інтерфейсу, яким його бачить кінцевий користувач системи. Це може бути як документ, що показує зовнішній вигляд екрану або структуру звіту, так і сам екран (звіт), створений за

допомогою одного із засобів візуальної розробки додатків - четвертого покоління 4GL - Fourth Generation Languages. Так як більшість мов 4GL дозволяють швидко створювати працюючі прототипи додатків, користувач має можливість побачити працюючий прототип системи на ранніх стадіях проектування.

Після створення підсхем реляційної моделі для додатків проектується детальна структура кожної програми у вигляді схеми навігації екранів, звітів, процедур пакетної обробки. На цьому кроці структура ОП деталізується до:

- вказівки конкретних стовпців і таблиць бази даних;
- правил їх обробки до вигляду екранних форм і звітів.

Отримана модель детально описується програмним кодом та безпосередньо використовується для програмування специфікованих інтерфейсів.

Далі за допомогою засобів розробки додатків відбувається фізичне створення системи: додатки програмуються і інтегруються в інформаційну систему.

3.1.2. Інструментальний засіб SE Companion

Інструментальний засіб SE Companion [7] є середовищем, в якому реалізований електронний варіант методології DATARUN. Він дозволяє:

- створити гіпертекстовий опис методології у вигляді ієрархії опису стадій, етапів і операцій розробки;
- створити гіпертекстовий опис усіх методів і методик реалізації процесів ЖЦ ПЗ;

- виділити з гіпертекстового опису ієрархію процесів ЖЦ ПЗ для планування та керування процесом створення ПЗ (ієрархію робіт);

- змінювати гіпертекстові описи ЖЦ до вигляду, необхідного розробнику, шляхом проведення авторизації методології і відслідковувати ці зміни в ієрархії робіт, які забезпечують керування проектом;

- прив'язати до процесів ЖЦ інструментальні засоби підтримки цих процесів і забезпечити виклик інструментальних засобів з відповідних екранів гіпертекстового довідника;

- забезпечити перегляд гіпертекстових екранів опису методів, що використовуються з інструментальних засобів;

- забезпечити підтримку процесу керування розробкою, зокрема, за рахунок взаємодії із засобом планування робіт MS Project, оцінювати трудомісткості проекту, відстежувати виконання робіт, створювати графіки робіт, тощо.

Особливо важливими є можливості авторизації методології та інтерактивний доступ будь-якого розробника до опису будь-якого методу або процесу в потрібний йому момент часу.

На сучасному етапі розвитку технології, в умовах швидкої зміни як програмних і апаратних засобів, так і задач бізнесу, методологія створення, супроводу і розвитку ПЗ не повинна бути статичною, вона повинна мати можливість динамічного розвитку та налаштування на нові технології, методи та інструментальні засоби.

Сучасні розробники великих ІС використовують одну або кілька методологій постачальника, а потім створюють на їх основі власні методології і технології, адаптовані до конкретних умов власної сфери діяльності(див. підрозділ 1.3).

У SE Companion вихідним документом, що описує методологію (як процеси ЖЦ, так і всі супутні методи і методики), є файл у форматі MS Word. Це забезпечує можливість:

- опису методології з будь-якою ступінню деталізації;

- проведення розмітки для створення гіпертексту;

- авторизації методології в прийнятому стандартному форматі.

Гіпертекстовий опис методології та технології створення ПЗ будується з:

- опису процесів життєвого циклу;

- методів і методик;

- представляє собою єдиний гіпертекстовий документ у форматі MS Help. Підсумковий гіпертекстовий опис формується в результаті трансляції вихідного документа.

Опис методології створення системи складається з розділу опису процесів ЖЦ і розділів опису методів і методик. У свою чергу, розділ описів процесів має наступну ієрархію:

- описів стадій;

- опис етапів і операцій життєвого циклу з обов'язковим описом вихідних компонентів кожного процесу.

Компоненти ПЗ створюються із застосуванням методик і методів, описаних у відповідних розділах.

Мінімальна конфігурація апаратно-програмних засобів, необхідних SE Companion Authoring Tool:

- процесор: i486/33;

- оперативна пам'ять: 4 Мбайт для перегляду гіпертексту, 12 Мбайт для авторизації;

- дискова пам'ять: 20 Мбайт;

- операційні середовища: родина Microsoft Windows.

3.2. CASE-засоби. Загальна характеристика і класифікація

Сучасні CASE-засоби охоплюють велику область підтримки чисельних технологій проектування ІС: від простих засобів аналізу і документування до повномасштабних засобів автоматизації, що покривають весь життєвий цикл ПЗ.

Найбільш трудомісткими етапами розробки ІС є етапи аналізу та проектування, у процесі яких CASE-засоби забезпечують якість прийнятих технічних рішень та підготовку проектної документації. При цьому велику роль відіграють методи візуального представлення інформації. Це передбачає

- побудову структурних чи інших діаграм у реальному масштабі часу,

- використання різноманітної кольорової палітри;

- наскрізну перевірку синтаксичних правил;

- використання графічних засобів моделювання предметної області, які дозволяють розробникам в наочному вигляді вивчати існуючу ІС, перебудовувати її у відповідності з поставленими цілями і наявними обмеженнями.

У розряд CASE-засобів потрапляють як відносно дешеві системи для персональних комп'ютерів (ПК) з дуже обмеженими можливостями, так і дорогі системи для неоднорідних обчислювальних платформ і операційних середовищ. Так, сучасний ринок програмних засобів налічує близько 300 різних типів CASE-

засобів, найбільш потужні з яких використовуються практично всіма провідними фірмами світової спільноти.

Зазвичай до CASE-засобів відносять будь-який програмний засіб, що автоматизує ту чи іншу сукупність процесів життєвого циклу ПЗ і володіє такими основними характерними особливостями:

- потужні графічні засоби для опису і документування ІС, що забезпечують зручний інтерфейс з розробником і розвивають його творчі можливості;

- інтеграція окремих компонентів серед CASE-засобів забезпечує керування процесом розробки ІС;

- використання спеціальним чином організованого сховища проектних метаданих.

Інтегрований CASE-засіб (або комплекс засобів, що підтримують повний ЖЦ ПЗ) містить наступні компоненти:

- репозиторій, що є основою CASE-засобів. Він повинен забезпечувати зберігання версій проекту і його окремих компонентів, синхронізацію надходження інформації від різних розробників при груповій розробці, контроль метаданих на повноту і несуперечність;

- графічні засоби аналізу і проектування, що забезпечують створення та редагування ієрархічно пов’язаних діаграм (DFD, ERD тощо), що утворюють моделі ІС;

- засоби розробки додатків, включаючи мови 4GL і генератори кодів;

- засоби конфігураційного керування;

- засоби документування;

- засоби тестування;

- засоби керування проектом;

- засоби реінженірингу.

Вимоги до функцій окремих компонентів у вигляді критеріїв оцінки CASE-засобів наведені в розділі 4.2.

Всі сучасні CASE-засоби класифікуються в основному за типами та категоріями.

Класифікація за типами відображає функціональну орієнтацію CASE-засобів на ті чи інші процеси ЖЦ ПЗ чи системи.

Класифікація за категоріями визначає ступінь інтегрованості по функціях, які виконуються, і включає окремі локальні засоби, що вирішують:

- невеликі автономні задачі (tools);

- набір частково інтегрованих засобів, що охоплюють більшість етапів життєвого циклу ІС (toolkit);

- повністю інтегровані засоби, які підтримують весь ЖЦ ІС і пов'язані спільним репозиторієм.

Крім цього, CASE-засоби можна класифікувати за наступними ознаками:

- застосовуваними методологіями і моделями систем і БД;

- ступенями інтеграції із СУБД;

- доступними платформами.

Класифікація за типами переважно збігається з компонентним складом CASE-засобів і включає наступні основні типи:

- засоби аналізу (Upper CASE), призначені для побудови та аналізу моделей предметної області (Design / IDEF (Meta Software), BPwin (Logic Works));

- засоби аналізу і проектування (Middle CASE), що підтримують найбільш поширені методології проектування і використовуються для створення проектних специфікацій (Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE. Аналітик (МакроПроджект)). Виходом засобів є специфікації компонентів і інтерфейсів системи, архітектура системи, алгоритмів і структур даних;

- засоби проектування баз даних для забезпечення моделювання даних і генерації схем баз даних (як правило, на мові SQL) для найбільш поширених СУБД. До них відносяться ERwin (Logic Works), S-Designor (SDP) і DataBase Designer (ORACLE). Засоби проектування баз даних є також у складі CASE-засобів Vantage Team Builder, Designer/2000, Silverrun і PRO-IV;

- засоби розробки додатків. До них відносяться засоби 4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/2000 (ORACLE), New Era (Informix), SQL Windows (Gupta), Delphi (Borland) тощо) і генератори кодів, що входять до складу Vantage Team Builder, PRO-IV і частково - в Silverrun;

- засоби реінженірингу, щоб забезпечити аналіз програмних кодів і схем баз даних і формування на їх основі різних моделей і проектних специфікацій. Засоби аналізу схем БД і формування ERD входять до складу Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin і S-Designor. У сфері аналізу програмних кодів найбільше поширення одержують об’єктно-орієнтовані CASE-засоби, що забезпечують реінжиніринг програм на мові C++ (Rational Rose, Rational Software), Object Team (Cayenne)).

Допоміжні типи включають:

- засоби планування та керування проектом (SE Companion, Microsoft Project та ін);

- засоби конфігураційного керування (PVCS (Intersolv));

- засоби тестування (Quality Works (Segue Software));

- засоби документування (SoDA (Rational Software)).

На сьогоднішній день ринок програмного забезпечення має такі найрозвинутіші CASE-засоби:

- Vantage Team Builder (Westmount I-CASE);

- Designer/2000;

- Silverrun;

- ERwin + BPwin;

- S-Designor;

- CASE.Аналітик.

Опис перерахованих CASE-засобів наведено в розділі 5. Крім того, на ринку постійно з'являються як нові для вітчизняних користувачів системи (наприклад, CASE / 4 / 0, PRO-IV, System Architect, Visible Analyst Workbench, EasyCASE), так і нові версії і модифікації перерахованих систем.

4. Технологія впровадження CASE-засобів

Наведена в даному розділі технологія базується в основному на стандартах IEEE [6,7] (IEEE - Institute of Electrical and Electronics Engineers - Інститут інженерів з електротехніки та електроніки). Термін "впровадження" використовується в широкому сенсі і включає всі дії: від оцінки первинних потреб до повномасштабного використання CASE-засобів в різних підрозділах організації-користувача. Процес впровадження CASE-засобів складається з наступних етапів [6]:

- визначення потреби у CASE-засобах;
- оцінка і вибір конфігурації CASE-засобів;
- виконання пілотного проекту;
- практичне впровадження CASE-засобів.

Процес успішного впровадження CASE-засобів не обмежується тільки їх використанням. Насправді він охоплює планування і реалізацію безлічі технічних, організаційних, структурних процесів, змін в загальній культурі організації і заснований на чіткому розумінні можливостей CASE-засобів.

На спосіб впровадження CASE-засобів може вплинути специфіка конкретної ситуації. Наприклад, якщо замовник віддає перевагу конкретному засобу, або він обумовлюється вимогами контракту, етапи впровадження повинні відповідати такому зумовленому вибору. В інших ситуаціях відносна простота або складність засобів, ступінь узгодженості або конфліктності з існуючими в організації процесами, необхідна ступінь інтеграції з іншими засобами, досвід і кваліфікація користувачів можуть

призвести до внесення відповідних коректив у процес впровадження.

4.1. Визначення потреб у CASE-засобах

Даний етап (рисунок 4.1.1) включає структуру досягнення розуміння потреби організації у використанні CASE-засобів і технологію подальшого процесу впровадження CASE-засобів. Він повинен призвести до виділення тих областей діяльності організації, в яких застосування CASE-засобів може принести реальну користь.

Результатом даного етапу є документ, що визначає стратегію впровадження CASE-засобів.

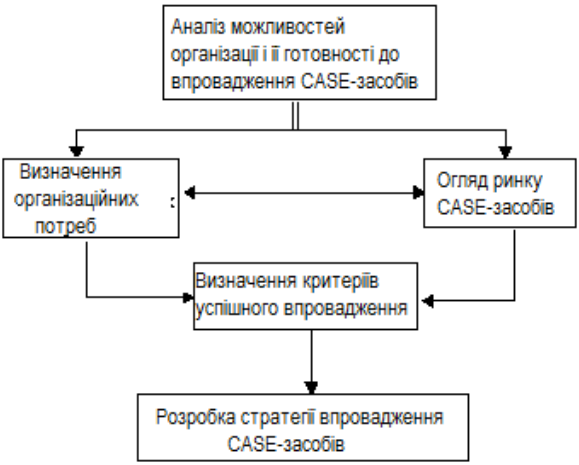

Рисунок 4.1.1 - Визначення потреб у CASE-засобах

4.1.1. Аналіз можливостей організації

Першою дією даного етапу є аналіз можливостей організації щодо її технологічної бази, персоналу і використовуваного ПЗ. Такий аналіз може бути формальним або неформальним.

Формальні підходи визначаються моделлю оцінки зрілості технологічних процесів організації CMM (Capability Maturity Model), розробленої SEI (Software Engineering Institute), а також стандартами ISO 9001:1994, ISO 9003-3:1991 та ISO 9004-2:1991. У центрі уваги цих підходів є аналіз різних аспектів, які відбуваються в організації процесів.

Для отримання інформації щодо стану та потреб організації можуть використовуватися неформальні оцінки та анкетування. Список простих питань, які можуть допомогти в неформальній оцінці поточної практики використання ПЗ, технології і персоналу, наведено нижче.

Відповіді на дані питання можуть визначити ті області, де автоматизація може принести певний економічний ефект. В іншому випадку може виявитися, що вдосконалення процесу розробки та супроводу ПЗ, програм навчання та інших функцій більш економічно привабливе, ніж придбання нових засобів. Деякі з цих удосконалень можуть виявитися необхідними для отримання максимальної вигоди від впровадження будь-яких засобів.

Дані питання є, по суті, керівництвом щодо збору інформації, необхідної для визначення ступеня готовності організації до впровадження CASE-технології.

Загальні питання:

- модель ЖЦ, що використовується (каскадна або спіральна);
- методи, що використовуються (структурні, об'єктно-орієнтовані). Ступінь адаптації методу до потреб організації;
- кваліфікація співробітників;
- наявність документованих стандартів (формальних чи неформальних) з аналізу вимог, специфікацій та проектування, кодування і тестування;
- кількісні метрики, що використовуються в процесі розробки ПЗ, їх використання;
- види документації, що випускається в процесі ЖЦ ПЗ;
- наявність групи підтримки засобів проектування.

Проекти, що ведуться в організації:

- середня тривалість проекту в людино-місяцях;
- середня кількість фахівців, що беруть участь в проектах різних категорій (невеликих, середніх і великих);
- середній розмір проектів різних категорій в термінах кодових метрик (наприклад, у рядках вихідних кодів), спосіб вимірювання.

Технологічна база. Технологічна база організації містить не лише технічні засоби, що використовуються при розробці ПЗ, але також включає мови програмування, засоби, методи і середовище функціонування ПЗ. Ця база дуже істотно впливає на вибір відповідних CASE-засобів. Питання, що стосуються технології, наступні:

- доступні обчислювальні ресурси, платформа розробки;
- рівень доступності ресурсів, вузькі місця, середній час очікування ресурсів;

- ПЗ, що використовується в організації, і його характер (готові програмні продукти, власні розробки);

- ступінь інтеграції програмних продуктів, що використовуються, механізми інтеграції (існуючі та плановані);

- тип і рівень мережевих можливостей, доступних групі розробників;

- мови програмування, що використовуються;

- середній відсоток розроблених нових, повторно використовуваних і реально експлуатованих програм.

Персонал. Головною метою оцінки персоналу є визначення його ставлення до можливих змін (позитивного, нейтрального чи негативного). Питання, що стосуються оцінки персоналу, включають наступні:

- реакція співробітників організації (як окремих людей, так і колективів) на впровадження нової технології. Наявність досвіду успішних або неуспішних впроваджень;

- наявність лідерів, здатних серйозно вплинути на ставлення до нових засобів розробки;

- наявність прагнення "знизу" до вдосконалення засобів і технології;

- обсяг навчання, необхідний для орієнтації користувачів у новій технології;

- стабільність і рівень плинності кадрів.

Готовність. Метою оцінки готовності організації є визначення того, наскільки вона здатна сприйняти як негайні, так і довготермінові наслідки впровадження CASE-засобів. Питання, що стосуються оцінки готовності, наступні:

- підтримка проекту з боку вищого керівництва;
- готовність організації до довготермінового фінансування проекту;
- готовність організації до виділення необхідних фахівців для участі в процесі впровадження і до їхнього навчання;
- готовність персоналу до істотної зміни технології своєї роботи;
- ступінь розуміння персоналом масштабу змін;
- готовність технічних фахівців і менеджерів піти на можливе короткочасне зниження продуктивності своєї роботи;
- готовність керівництва до довготривалого очікування віддачі від вкладених коштів.

Оцінка готовності організації до впровадження CASE-технології повинна бути відвертою і ретельною, оскільки в разі відсутності такої готовності всі зусилля з впровадження потерплять крах.

4.1.2. Визначення організаційних потреб

Організаційні потреби йдуть безпосередньо з проблем організації і цілей, які вона прагне досягти. Проблеми та цілі пов’язані з керуванням, виробництвом продукції, економікою, персоналом чи технологією. Питання, що стосуються визначення цілей, потреб та очікуваних результатів, наведені нижче.

Визначення потреб повинне виконуватися в поєднанні з оглядом ринку CASE-засобів, оскільки інформація про технології, доступні на ринку в даний момент, може вплинути на потреби.

Цілі організації. Цілі організації відіграють головну роль у визначенні її конкретних потреб та очікуваних результатів. Для їх розуміння необхідно відповісти на наступні питання:

- намір організації використовувати CASE-технологію для допомоги в досягненні певних цілей або очікувань (наприклад, певного рівня CMM або сертифікації відповідно до ISO 9001);

- сприйняття CASE-технології як чинника, що сприяє досягненню стратегічних цілей організації;

- наявність у організації власної програми вдосконалення процесу розробки ПЗ;

- сприйняття ініціативи впровадження CASE-технології, як частини більш широкомасштабного проекту по створенню середовища розробки ПЗ.

Потреби організації. Визначення потреб організації, пов'язаних з використанням CASE-технології, включає аналіз цілей і існуючих можливостей. Після того, як всі потреби організації визначені, кожній з них повинен бути привласнений певний пріоритет, що відображає її значимість для успішної діяльності. Якщо потреби, пов'язані з CASE-технологією, не одержать вищий пріоритет, є сенс відмовитися від її впровадження і зосередитися на потребах з найвищим пріоритетом.

Доцільно побудувати матрицю відповідності потреб організації з можливостями основних CASE-засобів. Складання такої матриці вимагає певного рівня знань ринку CASE-засобів. У

кінцевому рахунку, кожна функція або можливість засобу повинна точно відповідати деякій потребі з певним пріоритетом.

Визначенню потреб організації можуть допомогти відповіді на наступні питання:

- яким чином продуктивність і якість діяльності організації порівнюються з аналогічними показниками подібних організацій (на жаль, багато організацій не мають даних для такого порівняння);

- які процеси ЖЦ ПЗ дають найкращу (і відповідно, найгіршу) віддачу;

- чи існують конкретні процеси, які можуть бути вдосконалені шляхом використання нових методів і засобів.

Очікувані результати. З впровадженням CASE-засобів зазвичай пов'язують великі очікування. У ряді випадків ці очікування виявляються нереалістичними і призводять до невдачі при впровадженні.

Складання реалістичного переліку очікуваних результатів є важким завданням, оскільки він може залежати від таких факторів, як тип засобів, що впроваджуються, і характеристики організації.

Ряд потенційно реалістичних і нереалістичних очікуваних результатів, пов'язаних:

- з організацією в цілому;
- користувачами;
- плануванням;
- аналізом;
- проектуванням;
- розробкою та витратами.

Практично неможливо, щоб в процесі одного впровадження CASE-засобів були досягнуті всі позитивні результати. Тим не менш, будь-яка організація може виробити власний підхід до очікуваних результатів, маючи на увазі, що даний перелік є всього лише прикладом.

Реалістичні очікування:

- підвищення уваги до планування діяльності, пов'язаної з інформаційною технологією;
- підтримка реінженірингу бізнес-процесів;
- довготривале підвищення продуктивності і якості діяльності організації;
- прискорення і підвищення узгодженості розробки додатків;
- зниження частки ручної праці в процесі розробки та/або експлуатації;
- більш точну відповідність програм до вимог користувачів;
- відсутність необхідності великої переробки програм для підвищення їх ефективності;
- поліпшення реакції служби експлуатації на вимоги внесення змін і вдосконалень;
- підвищення якості документування;
- поліпшення комунікації між користувачами та розробниками;
- послідовне і постійне підвищення якості проектування;
- більш високі можливості повторного використання розробок;
- короткочасне зростання витрат, пов'язане з діяльністю по впровадженню CASE-засобів;
- послідовне зниження загальних витрат;

- поліпшення прогнозованості витрат.

Нереалістичні очікування:

- відсутність впливу на загальну культуру і розподіл ролей в організації;

- розуміння проектних специфікацій непідготовленими користувачами;

- скорочення персоналу, пов'язане з впровадженням інформаційної технології;

- зменшення ступені участі в проектах вищого керівництва і менеджерів, а також експертів предметної області, зменшення ступені участі користувачів у процесі розробки програм;

- негайне підвищення продуктивності праці та інших показників діяльності організації;

- досягнення абсолютної повноти і несуперечності специфікацій;

- автоматична генерація прикладних систем з проектними специфікаціями;

- негайне зниження витрат, пов'язаних з інформаційною технологією;

- зниження витрат на навчання.

Реалізм в оцінці очікуваних витрат має особливо важливе значення, оскільки він дозволяє правильно оцінити віддачу від інвестицій. Витрати на впровадження CASE-засобів звичайно недооцінюються. Серед конкретних статей витрат на впровадження можна виділити наступні:

- фахівці з планування впровадження CASE-засобів;

- вибір, встановлення та облік специфічних вимог до персоналу;

- придбання CASE-засобів та навчання персоналу;

- налаштування;

- підготовка документації, стандартів і процедур використання засобів;

- інтеграція з іншими засобами та існуючими даними в організації-замовнику;

- освоєння засобів розробниками;

- технічні засоби;

- оновлення версій.

Важливо також усвідомлювати, що поліпшення діяльності організації, що є наслідком використання CASE-технології, може бути неочевидним протягом найпершого проекту, що використовує нову технологію. Продуктивність та інші характеристики діяльності організації можуть спочатку навіть погіршитися, оскільки на освоєння нових засобів та внесення необхідних змін у процес розробки потрібен деякий час. Таким чином, очікувані результати повинні розглядатися з урахуванням імовірного відтермінування в поліпшенні проектних характеристик.

Кожна потреба повинна мати певний пріоритет, що залежить від того, наскільки критичною вона є для досягнення успіху в організації. У кінцевому рахунку повинен чітко простежуватися вплив кожної функції або можливості придбаних засобів на задоволення конкретних потреб.

Результатом даної дії є формулювання потреб з їх пріоритетами, які використовуються на етапі оцінки і вибору в якості " потреб для користувача ".

4.1.3. Аналіз ринку CASE-засобів

Потреби організації в CASE-засобах повинні не конфліктувати з реальною ситуацією на ринку програмних продуктів або власними можливостями розробки. Дослідження ринку проводиться шляхом вивчення літератури з CASE-засобів, відвідування конференцій та семінарів, що проводяться постачальниками і користувачами CASE-засобів.

При проведенні даного аналізу необхідно з'ясувати можливість інтеграції конкретного CASE-засобу з іншими засобами, що використовуються (або запланованими до використання) організацією. Крім того, важливо отримати достовірну інформацію про засоби розробки, засновані на реальному користувацькому досвіді і відомостях від користувацьких груп.

4.1.4. Визначення критеріїв успішного впровадження

Зумовлені критерії повинні дозволяти кількісно оцінювати ступінь задоволення потреб, пов'язаних з впровадженням. Крім того, за кожним критерієм має бути визначено його конкретне оптимальне значення. На певних етапах впровадження ці критерії повинні аналізуватися для того, щоб визначити поточну ступінь задоволення потреб.

Як правило, більшість організацій здійснює впровадження CASE-засобів для того, щоб підвищити продуктивність процесів розробки та супроводу ПЗ, а також якість результатів розробки. Однак, ряд організацій не займаються і не займалися раніше збором

кількісних даних за вказаними параметрами. Відсутність таких даних ускладнює кількісну оцінку впливу, що чиниться впровадженням CASE-засобів. У цьому випадку рекомендується розробка відповідних метрик. Інформація про такі метрики приведена в стандартах IEEE Std 1045-1992 (IEEE Standard for Software Productivity Metrics) і IEEE Std 1061-1992 (IEEE Standard for a Software Quality Metrics Methodology).

У тому випадку, якщо базові метричні дані відсутні, організація часто може отримати корисну інформацію зі своїх проектних архівів.

Крім продуктивності та якості, корисну інформацію про стан впровадження CASE-засобів також можуть дати і інші характеристики організаційних процесів і персоналу. Наприклад, оцінка ступені успішності впровадження може включати відсоток проектів, що використовують CASE-засоби, рейтингові оцінки рівня кваліфікації фахівців, пов’язані з використанням CASE-засобів і результати опитувань персоналу з приводу ставлення до використання CASE-засобів. Інші приклади проектних характеристик, які можуть бути оцінені кількісно, включають наступні:

- узгодженість проектних результатів;
- точність вартісних і планових оцінок;
- мінливість зовнішніх вимог;
- дотримання стандартів організації;
- ступінь повторного використання існуючих компонентів ПЗ;
- обсяг і види необхідного навчання;
- типи і моменти виявлення проектних помилок;

- обчислювальні ресурси, які використовуються CASE-засобами.

4.1.5. Розробка стратегії впровадження CASE-засобів

Стратегія впровадження повинна забезпечувати задоволення потреб і критеріїв, визначених раніше. Стратегія включає наступні складові:

- організаційні потреби;
- базові метрики, необхідні для подальшого порівняння результатів;
- критерії успішного впровадження, пов'язані із задоволенням організаційних потреб, включаючи очікувані результати послідовних етапів процесу впровадження;
- підрозділи організації, в яких має виконуватися впровадження CASE-засобів;
- вплив, який чиниться на інші підрозділи організації;
- стратегії і плани оцінки та вибору, пілотного проектування і переходу до повномасштабного впровадження;
- основні фактори ризику;
- орієнтовний рівень витрат і джерела фінансування процесу впровадження CASE-засобів;
- ключовий персонал та інші ресурси.

Необхідно зазначити, що впровадження нової технології може включати важливі і складні зміни в культурі організації. Істотна увага має приділятися ролям різних груп, залучених до процесу таких змін. Найбільш істотні ролі включають наступні:

- спонсор (звичайно з числа менеджерів вищого рівня). Дана роль є критичною для підтримки проекту і забезпечення необхідного фінансування. Спонсор повинен володіти чітким розумінням необхідності серйозних зусиль, пов'язаних з впровадженням CASE-засобів, і тривалості періоду очікування відчутних результатів;

- виконавець - зазвичай особа (або група осіб), яка усвідомлює потенційні можливості нової технології, яка користується авторитетом серед технічного персоналу і здатна очолити процес впровадження нової технології;

- цільова група - звичайно включає менеджерів і технічний персонал, які будуть залучені до безпосереднього використання CASE-засобів, а також спеціалістів, які будуть залучені побічно, таких, як фахівці з документування, персонал підтримки мережі та замовники. Повинні бути визначені потреби кожної з таких груп і план їх ефективного задоволення.

У загальному випадку впровадження CASE-засобів має керуватися і фінансуватися таким же чином, як і будь-який проект розробки ПЗ. Стратегія впровадження може бути переглянута у разі появи додаткової інформації.

Існує кілька підходів до розробки стратегії впровадження CASE-засобів. Відносні переваги того чи іншого підходу перед іншими повинні розглядатися в контексті специфіки конкретної організації. Особливе значення при цьому надається персоналу організації і процесу розробки ПЗ.

Спадний підхід до розробки стратегії визнає важливість дослідження всіх типів CASE-засобів і документування процесів розробки і супроводу ПЗ в даній організації до того, як

визначаються вимоги до CASE-засобів. При цьому виконується загальний аналіз процесу створення і супроводу ПЗ в організації. Даний підхід найчастіше тягне за собою загальну реорганізацію процесів створення і супроводу ПЗ в тій мірі, в якій це пов'язано з CASE-засобами. Результатом такої реорганізації стає великомасштабна стратегія автоматизації процесів створення і супроводу ПЗ.

Перевага низхідного підходу полягає в тому, що він охоплює всі процеси створення і супроводу ПЗ, забезпечуючи максимально можливу їх автоматизацію. Іншою перевагою є придбання інтегрованого набору засобів, оскільки кожна окрема поставка підпорядковується загальній стратегії. Спадний підхід також може бути легко інтегрований в загальну стратегію розвитку процесу створення і супроводу ПЗ, в якій впровадження CASE-засобів є тільки одним з аспектів.

Недоліки даного підходу полягають в наступному:

- спадний підхід вимагає для своєї реалізації значних людських і фінансових ресурсів;

- у загальному випадку, широкомасштабний підхід такого роду не дозволяє користувачам досить швидко приступити до практичного використання засобів;

- спадний підхід може призвести до відносно серйозних змін існуючих в організації процесів. Реалізацією такого підходу складніше скеровувати і, крім того, він містить в собі підвищений ризик провалу, що веде до того, що CASE-засоби "кладуться на полицю".

Спадний підхід рекомендується для відносно зрілих організацій із сталим процесом створення і супроводу ПЗ, які

прагнуть вкласти всі необхідні ресурси в повністю закінчену роботу. Щоб підвищити імовірність успіху, необхідне прийняття серйозних зобов'язань з боку як керівництва, так і потенційних користувачів.

Висхідний підхід починається з визначення деякого засобу або типу засобів, які потенційно можуть допомогти організації в поліпшенні виконання поточної роботи. Організація може потім оцінити можливий вплив засобів на процес розробки і супроводу ПЗ.

Переваги даного підходу полягають в наступному:

- невелика автоматизація може бути виконана при мінімальних витратах;

- автоматизація може бути виконана за короткий проміжок часу, дозволяючи швидко усунути відомі недоліки в існуючих процесах;

- невеликий масштаб висхідної стратегії дозволяє краще фокусувати і контролювати вплив, який чиниться на існуючі процеси.

Недоліки даного підходу полягають в наступному:

- засоби, придбані як результат окремо взятих застосувань даного підходу, можуть погано інтегруватися між собою. Це може призвести до необхідності виконання великого обсягу ручної роботи;

- в той час, як конкретні, порівняно невеликі проблеми вирішуються досить швидко, до вирішення фундаментальних проблем, пов'язаних з широким колом процесів розробки ПЗ, справа зазвичай не доходить.

Висхідний підхід рекомендується для організацій з вузько специфічними потребами в автоматизації, не потребуючи загальне вдосконалення процесів. У деяких випадках може виявитися не занадто практичним приступати до такого вдосконалення, не визначивши найбільш нагальні потреби в автоматизації. У той час як даний підхід може допомогти організації задовольнити нагальні потреби і розвинути основні процеси, залишається суттєва небезпека того, що обраний засіб не зробить істотного впливу на такі фактори, як якість і продуктивність.

Найбільш раціональна стратегія може поєднувати характеристики обох підходів. Наприклад, спадні методи можуть використовуватися для визначення стандартів якості організації, потреб у засобах і очікуваних результатах, тоді як висхідні методи можуть використовуватися для оцінки й вибору конкретних CASE-засобів, розробки планів впровадження та контролю його результатів.

4.2. Оцінка і вибір CASE-засобів

4.2.1. Загальні відомості

Модель процесу оцінки та вибору CASE-засобів [7], розглянута нижче (рисунок 4.2.1.1), описує найбільш загальну ситуацію оцінки і вибору, а також показує залежність між ними. Як можна бачити, оцінка і вибір можуть виконуватися незалежно один від одного або разом, кожен з цих процесів вимагає застосування певних критеріїв.

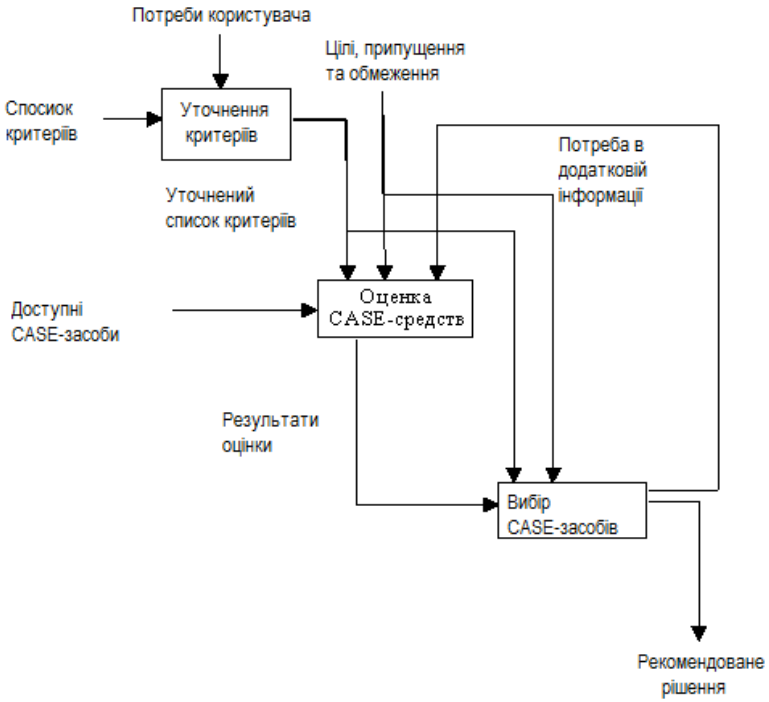

Рисунок 4.2.1.1 - Модель процесу оцінки і вибору

Процес оцінки і вибору може переслідувати декілька цілей, включаючи одну чи більше:

- оцінка декількох CASE-засобів і вибір одного або більше з них;

- оцінка одного або більше CASE-засобів і збереження результатів для подальшого використання;

- вибір одного або більше CASE-засобів з використанням результатів попередніх оцінок.

Як видно з рисунку, вхідною інформацією для процесу оцінки є:

- визначення користувацьких потреб;

- цілі та обмеження проекту;

- дані про доступні CASE-засоби;

- список критеріїв, які використовуються в процесі оцінки.

Результати оцінки можуть включати результати попередніх оцінок. При цьому не слід забувати, що набір критеріїв, що використовувалися при попередній оцінці, повинен бути сумісним з поточним набором. Конкретний варіант реалізації процесу (оцінка і вибір, оцінка для майбутнього вибору або вибір, заснований на попередніх оцінках) визначається перерахованими вище цілями.

Елементи процесу включають:

- цілі, припущення та обмеження, які можуть уточнюватися в ході процесу;

- потреби користувачів, що відображають кількісні та якісні вимоги користувачів до CASE-засобів;

- критерії, що визначають набір параметрів, відповідно до яких проводиться оцінка і прийняття рішення про вибір;

- формалізовані результати оцінок одного або більше засобів;

- рекомендоване рішення (або рішення про вибір, або подальша оцінка).

Процес оцінки і/або вибору може бути розпочатий тільки тоді, коли особа, група або організація повністю визначила для себе конкретні потреби і формалізувала їх у вигляді кількісних і якісних вимог у заданій предметній області. Термін "користувацькі вимоги" далі означає саме такі формалізовані вимоги.

Користувач повинен визначити конкретний порядок дій і прийняття рішень з будь-якими необхідними ітераціями. Наприклад, процес може бути представлений у вигляді дерева

рішень з його послідовним обходом і вибором підмножин кандидатів для більш детальної оцінки. Опис послідовності дій має визначати потік даних між ними.

Визначення списку критеріїв засновано на користувацьких вимогах і включає:

- вибір критеріїв для використання з наведеного далі переліку;
- визначення додаткових критеріїв;
- визначення області використання кожного критерію (оцінка, вибір або обидва процеси);
- визначення однієї або більше метрик для кожного критерію оцінки;
- призначення ваги кожному критерію при виборі.

4.2.2. Процес оцінки

Метою процесу оцінки є визначення функціональності і якості CASE-засобів для подальшого вибору. Оцінка виконується відповідно до конкретних критеріїв, її результати включають як об'єктивні, так і суб'єктивні дані по кожному засобу.

Процес оцінки включає наступні дії:

- формулювання задачі оцінки, включаючи інформацію про мету і масштаби оцінки;

- визначення критерію оцінки, що випливає з визначення завдання;

- визначення засобів-кандидатів шляхом перегляду списку кандидатів і аналізу інформації про конкретні засоби;

- оцінка засобів-кандидатів у контексті обраних критеріїв. Необхідні для цього дані можуть бути отримані шляхом аналізу самих засобів і їх документування, опитування користувачів, роботи з демоверсіями, виконання тестових прикладів, експериментального застосування засобів і аналізу результатів попередніх оцінок, підготовка звіту за результатами оцінки.

Одним з найважливіших критеріїв у процесі оцінки може бути потенційна можливість інтеграції кожного з засобів-кандидатів з іншими засобами, які вже знаходяться в експлуатації або плановані до використання в даній організації.

Масштаб оцінки повинен встановлювати необхідний рівень деталізації, необхідні ресурси та ступінь придатності її результатів. Наприклад, оцінка повинна виконуватися для набору з одного або більше конкретних CASE-засобів, що підтримують один чи більше

конкретних процесів створення і супроводу ПЗ або CASE-засобів, що підтримують один чи більше проектів або типів проектів.

Список CASE-засобів, можливих кандидатів, формується з різних джерел: оглядів ринку ПЗ, інформації постачальників, оглядів CASE-засобів і інших подібних публікацій.

Наступним кроком є отримання інформації про CASE-засоби, або отримання їх самих, або і те, і інше. Ця інформація може складатися з оцінок незалежних експертів, повідомлень і звітів постачальників CASE-засобів, результатів демонстрації можливостей CASE-засобів з боку постачальників та інформації, отриманої безпосередньо від реальних користувачів. Самі CASE-засоби можуть бути отримані шляхом придбання у вигляді оціночної копії або іншими методами.

Оцінка та накопичення даних може виконуватися наступними способами:

- аналіз CASE-засобів і документації постачальника;
- опитування реальних користувачів;
- аналіз результатів проектів, які використовували дані CASE-засоби;
- перегляд демонстрацій і опитування демонстраторів;
- виконання тестових прикладів;
- застосування CASE-засобів у пілотних проектах;
- аналіз будь-яких доступних результатів попередніх оцінок.

Існують як об’єктивні, так і суб’єктивні критерії. Результати оцінки відповідно до конкретного критерію можуть бути двійковими, знаходитися в деякому числовому діапазоні, являти собою просто числове значення або мати будь-яку іншу форму.

Для об’єктивних критеріїв оцінка повинна виконуватися шляхом відтворення процедури, щоб будь-який інший фахівець, що виконує оцінку, міг отримати такі ж результати. Якщо використовуються тестові приклади, їх набір повинен бути заздалегідь визначений, уніфікований і документований.

За суб’єктивними критеріями CASE-засіб повинен оцінюватися групою фахівців, що використовують одні й ті самі критерії. Необхідний рівень досвіду фахівців або груп повинен бути заздалегідь визначений.

Результати оцінки повинні бути стандартним чином документовані (для полегшення подальшого використання) і, при необхідності, затверджені.

Звіт за результатами оцінки повинен містити наступну інформацію:

- введення. Загальний огляд процесу і перелік основних результатів;

- передумови. Мета оцінки і бажані результати, період часу, протягом якого виконувалася оцінка, визначення ролей і відповідного досвіду фахівців, що виконували оцінку;

- підхід до оцінки. Опис загального підходу, включаючи отримані CASE-засоби; інформацію, що визначає контекст і масштаб оцінки, а також будь-які припущення й обмеження;

- інформація про CASE-засоби. Вона повинна включати наступне: найменування CASE-засобу, версію CASE-засобу, дані про постачальника, включаючи контактну адресу і телефон, конфігурацію технічних засобів, вартісні дані, опис CASE-засобу, що включає підтримувані даним засобом процеси створення і супроводу ПЗ, програмне середовище CASE-засобу (зокрема,

підтримувані мови програмування, операційні системи, сумісність з базами даних), функції CASE-засобу, вхідні/вихідні дані і область застосування;

- етапи оцінки. Конкретні дії, що виконуються в процесі оцінки, повинні бути описані зі ступінню деталізації, необхідної як для розуміння масштабу і глибини оцінки, так і для її повторення при необхідності;

- конкретні результати. Результати оцінки повинні бути представлені в термінах критеріїв оцінки. У тих випадках, коли звіт охоплює цілий ряд CASE-засобів або результати даної оцінки будуть зіставлятися з аналогічними результатами інших оцінок, необхідно звернути особливу увагу на формат представлення результатів, що сприяє такому порівнянню. Суб'єктивні результати повинні бути відокремлені від об'єктивних і повинні супроводжуватися необхідними поясненнями;

- висновки;

- програми. Формулювання задачі оцінки і уточнений список критеріїв.

4.2.3. Процес вибору

Процеси оцінки і вибору тісно взаємозалежні один з одним. За результатами оцінки, цілі вибору і/або критерію вибору і їх ваги, можуть вимагати модифікації. У таких випадках може знадобитися повторна оцінка.

Коли аналізуються остаточні результати оцінки і до них застосовуються критерії вибору, може бути рекомендовано придбання CASE-засобу або набору CASE-засобів з наявних.

Альтернативою може бути відсутність адекватних CASE-засобів, в цьому випадку рекомендується розробити новий CASE-засіб, модифікувати існуючий або відмовитися від впровадження цієї технології взагалі.

Процес вибору тісно взаємозалежний з процесом оцінки і включає наступні дії:

- формулювання задач вибору, включаючи мету, припущення й обмеження;

- виконання всіх необхідних дій по вибору, включаючи визначення і ранжування критеріїв, визначення засобів-кандидатів, збір необхідних даних та застосування ранжированих критеріїв до результатів оцінки для визначення засобів з найкращими показниками. Для багатьох користувачів важливим критерієм вибору є інтегрованість CASE-засобу з існуючим середовищем, а саме:

- виконання необхідної кількості ітерацій з тим, щоб вибрати (або відкинути) засоби, які мають схожі показники;

- підготовка звіту за результатами вибору.

У процесі вибору можливе одержання двох результатів:

- рекомендацій щодо вибору конкретного CASE-засобу;

- запиту на отримання додаткової інформації для процесу оцінки.

Масштаб вибору повинен встановлювати необхідний рівень деталізації, необхідні ресурси, графік і очікувані результати. Існує ряд параметрів, які можуть бути використані для визначення масштабу, включаючи:

- використання попереднього відбору (наприклад, відбір тільки засобів, що працюють на конкретній платформі);

- використання раніше отриманих результатів оцінки, результатів оцінки з зовнішніх джерел або комбінації того й іншого.

У тому випадку, якщо попередні оцінки виконувалися з використанням різних наборів критеріїв або виконувалися з використанням конкретних критеріїв, але різними способами, результати оцінок повинні бути представлені в узгодженій формі. Після завершення даного кроку, оцінка кожного CASE-засобу повинна бути представлена в рамках єдиного набору критеріїв і повинна бути безпосередньо порівняна з іншими оцінками.

Алгоритми, які зазвичай використовуються для вибору, можуть бути засновані на масштабі або ранзі.

Алгоритми, засновані на масштабі, обчислюють єдине значення для кожного CASE-засобу шляхом множення ваги кожного критерію на його значення (з урахуванням масштабу) і складання всіх утворень. CASE-засіб із найвищим результатом отримує перший ранг.

Алгоритми, засновані на ранзі, використовують ранжування CASE-засобів - кандидатів за окремими критеріями або групам критеріїв у відповідності зі значеннями останніх у заданому масштабі. Потім, аналогічно попередньому, ранги зводяться разом і обчислюються загальні значення рангів.

При аналізі результатів вибору передбачається, що процес вибору завершено, CASE-засіб вибрано і рекомендовано до використання. Але може знадобитися більш точний аналіз для визначення ступені залежності значень ключових критеріїв від відмінностей у значеннях характеристик CASE-засобів - кандидатів. Такий аналіз дозволить визначити, наскільки результат

ранжування CASE-засобів залежить від оптимальності вибору вагових коефіцієнтів критеріїв. Він також може використовуватися для визначення суттєвих відмінностей між CASE-засобами з дуже близькими значеннями критеріїв або рангів.

Якщо жоден з CASE-засобів не задовольняє мінімальних критеріїв, вибір (можливо, разом з оцінкою) може бути повторено для інших CASE-засобів - кандидатів.

Якщо відмінності між найкращими кандидатами несуттєві, додаткова інформація може бути отримана шляхом повторного вибору (можливо, разом з оцінкою) з використанням додаткових або інших критеріїв.

Рекомендації по вибору повинні бути суворо обґрунтовані. У разі відсутності адекватних CASE-засобів, як було зазначено вище, рекомендується розробити новий CASE-засіб, модифікувати існуючий або відмовитися від впровадження.

4.2.4 Критерії оцінки і вибору

Критерії формують базис для процесів оцінки і вибору і можуть приймати різні форми, включаючи:

- числові міри в широкому діапазоні значень, наприклад, обсяг необхідної пам'яті;

- числові міри в обмеженому діапазоні значень, наприклад, простота освоєння, виражена в балах від 1 до 5;

- виконавчі заходи (істина/хибність, так/ні), наприклад, здатність генерації документації у форматі Postscript;

- заходи, які можуть приймати одне або більше з кінцевих множин значень, наприклад, платформи, для яких підтримується CASE-засіб.

Типовий процес оцінки і/або вибору може використовувати набір критеріїв різних типів.

Структура набору критеріїв наведена на рисунку 4.2.4.1.

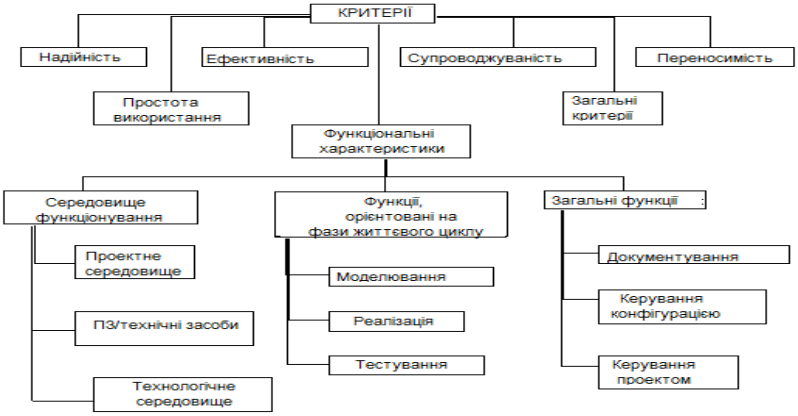

Рисунок 4.2.4.1 - Структура набору критеріїв

Кожен критерій має бути обраний і адаптований експертом з урахуванням особливостей конкретного процесу. У більшості випадків тільки деякі з безлічі описаних нижче критеріїв виявляються прийнятними для використання, при цьому також додаються додаткові критерії. Вибір і уточнення набору використовуваних критеріїв є критичним кроком у процесі оцінки і/або вибору.

Функціональні характеристики. Критерії першого класу призначені для визначення функціональних характеристик CASE-засобів. Вони в свою чергу поділяються на групи та підгрупи.

1. Середовище функціонування:

a. Проектне середовище:

- підтримка процесів життєвого циклу. Визначає набір процесів ЖЦ, які підтримує CASE-засіб. Прикладами таких процесів є аналіз вимог, проектування, реалізація, тестування і оцінка, супровід, забезпечення якості, керування конфігурацією і проектом, причому, вони залежать від прийнятої користувачем моделі ЖЦ;

- область застосування. Прикладами є системи обробки транзакцій, системи реального часу, інформаційні системи і т.д.;

- розмір, що підтримується. Визначає обмеження на такі величини, як кількість рядків коду, рівнів вкладеності, розмір бази даних, кількість елементів даних, кількість об'єктів конфігураційного управління.

b. ПЗ/технічні засоби:

- необхідні технічні засоби. Обладнання, необхідне для функціонування CASE-засобу, включаючи тип процесора, об'єм оперативної і дискової пам'яті;

- технічні засоби, що підтримуються. Елементи обладнання, які можуть використовуватися CASE-засобом, наприклад, пристрої введення/виведення;

- необхідне ПЗ. ПЗ, необхідне для функціонування CASE-засобу, включаючи операційні системи та графічні оболонки;

- ПЗ, що підтримується. Програмні продукти, які можуть використовуватися CASE-засобом.

Технологічне середовище:

- відповідність стандартам технологічного середовища. Такі стандарти стосуються мови програмування, бази даних, сховища, комунікацій, графічного інтерфейсу користувача, документації, розробки, керування конфігурацією, безпеки, стандартів обміну інформацією та інтеграції за даними, керуванням за призначеним для користувача інтерфейсом;

- сумісність з іншими засобами. Здатність до взаємодії з іншими засобами, включаючи безпосередній обмін даними (прикладами таких засобів є текстові процесори, бази даних та інші CASE-засоби). Можливість перетворення репозиторію або його частини в стандартний формат для обробки іншими засобами;

- методологія, що підтримується. Набір методів і методик, які підтримуються CASE-засобом. Прикладами є структурний або об'єктно-орієнтований аналіз та проектування;

- мови, що підтримуються. Всі мови, використовувані CASE-засобом. Прикладами таких мов є мови програмування, мови опису баз даних і мови запитів (DDL, SQL), графічні мови (Postscript, HPGL), мови специфікації проектних вимог і інтерфейси операційних систем (мови керування завданнями).

2. Функції, орієнтовані на фази життєвого циклу:

а. Моделювання:

Дані критерії визначають здатність виконання функцій, необхідних для формування специфікації вимог до ПЗ і перетворенню їх у проект:

- побудова діаграм. Можливість створення і редагування діаграм різних типів, що представляють цікавість для користувача. Найбільш поширені типи діаграм описані в розділі 2;

- графічний аналіз. Можливість аналізу графічних об’єктів, а також зберігання та подання проектної інформації в графічному вигляді. У більшості випадків графічні аналізатори інтегровані з засобами побудови діаграм;

- введення і редагування специфікацій вимог і проектних специфікацій. До специфікацій такого роду відносяться описи функцій, даних, інтерфейсів, структури, якості, продуктивності, технічних засобів, середовища, витрат і графіків;

- мова опису специфікації вимог і проектних специфікацій. Можливість імпорту, експорту і редагування специфікацій з використанням формальної мови;

- моделювання даних. Можливість введення і редагування інформації, яка описує елементи даних системи та їх відношення;

- моделювання процесів. Можливість введення і редагування інформації, яка описує процеси системи і їх відношення;

- проектування архітектури ПЗ. Проектування логічної структури ПЗ - структури модулів, інтерфейсів тощо;

- імітаційне моделювання. Можливість динамічного моделювання різних аспектів функціонування системи на основі специфікацій вимог і/або проектних специфікацій, включаючи зовнішній інтерфейс і продуктивність (наприклад, час відгуку, коефіцієнт використання ресурсів і пропускну здатність);

- прототипування. Можливість проектування і генерації попереднього варіанту всієї системи або її окремих компонентів на основі специфікацій вимог і/або проектних специфікацій. Прототипування в основному стосується зовнішнього інтерфейсу і здійснюється при безпосередній участі користувачів;

- генерація екранних форм. Можливість генерації екранних форм на основі специфікацій вимог і/або проектних специфікацій;

- можливість трасування. Можливість наскрізного аналізу функціонування системи від специфікації вимог до кінцевих результатів (встановлення і відстеження відповідностей та зв'язків між функціональними та іншими зовнішніми вимогами до ІС, технічними рішеннями і результатами проектування). Розрізняють пряме трасування (перевірка обліку всіх вимог) та зворотне трасування (пошук проектних рішень, не пов'язаних ні з якими зовнішніми вимогами);

- синтаксичний і семантичний контроль проектних специфікацій. Контроль синтаксису діаграм і типів їх елементів, контроль декомпозиції функцій, перевірка специфікацій на повноту і несуперечність;

- інші види аналізу. Конкретні додаткові види аналізу можуть включати алгоритми, потоки даних, формалізацію даних, використання даних, призначений для користувача інтерфейс;

- автоматизоване проектування звітів а вимогою користувачів а будь-який термін роботи системи.

b. Реалізація. Реалізація дозволяє створити функції, пов'язані з розробкою елементів системи (програмних кодів) або провести модифікацію існуючої системи. Більшість з перерахованих нижче критеріїв залежить від конкретних програмних кодів та мов програмування і включають наступні функції:

- синтаксично кероване редагування. Можливість введення і редагування вихідних кодів на одній або декількох мовах програмування з одночасним синтаксичним контролем;

- генерація коду. Можливість генерації кодів на одній або декількох мовах програмування на основі проектних специфікацій. Типи генерованого коду можуть включати звичайний програмний код, схему бази даних, запити, екрани/меню.

- компіляція коду;

- конвертування вихідного коду. Можливість перетворення коду з однієї мови в іншу;

- аналіз надійності. Можливість кількісно оцінювати параметри надійності ПЗ, такі, як кількість помилок тощо;

- реверсний інжиніринг. Можливість аналізу існуючих вихідних кодів і формування на їх основі проектних специфікацій;

- реструктуризація вихідного коду. Можливість модифікації формату і/або структури існуючого вихідного коду;

- аналіз вихідного коду. Прикладами такого аналізу можуть бути визначення розміру коду, обчислення показників складності, генерація перехресних посилань і перевірка на відповідність стандартам;

- налагодження. Типові функції налагодження - трасування програм, виділення вузьких місць і найбільш часто використовуваних фрагментів коду і т.д.;

с. Тестування. Критерії тестування включають наступні:

- опис тестів. Типові можливості включають генерацію тестових даних, алгоритмів тестування, необхідних результатів і т.д.;

- фіксація і повторення дій оператора. Можливість фіксувати дані, що вводяться оператором за допомогою клавіатури, миші і т.д., редагувати їх і відтворювати в тестових прикладах;

- автоматичний запуск тестових прикладів;

- регресивне тестування. Можливість повторення і модифікації раніше виконаних тестів для визначення відмінностей в системі і/або середовищі;

- автоматизований аналіз результатів тестування. Типові можливості включають порівняння очікуваних і реальних результатів, порівняння файлів, статистичний аналіз результатів тощо;

- аналіз тестового покриття. Оснащеність засобами контролю вихідного коду і аналіз тестового покриття. Перевіряються, зокрема, звернення до операторів, процедур і змінних;

- аналіз продуктивності. Можливість аналізу продуктивності програм. Аналізовані параметри продуктивності можуть включати використання центрального процесора, пам'яті, звернення до певних елементів даних і/або сегментів коду, часові характеристики тощо;

- аналіз виняткових ситуацій, які виникають у процесі тестування;

- динамічне моделювання середовища. Зокрема, можливість автоматично генерувати модельовані вхідні дані системи.

3. Загальні функції. Наведені нижче критерії визначають функції CASE-засобів, що охоплюють всю сукупність фаз ЖЦ. Підтримка всіх цих функцій здійснюється за допомогою репозиторію.

а. Документування:

- редагування текстів і графіки. Можливість вводити і редагувати дані в текстовому і графічному форматі;

- редагування за допомогою форм. Можливість підтримувати форми, визначені користувачами, вводити і редагувати дані відповідно з формами;

- можливості видавничих систем;

- підтримка функцій і форматів гіпертексту;

- відповідність стандартам документування;

- автоматичне вилучення даних з репозиторію;

- генерація документації за специфікаціями користувача.

b. Керування конфігурацією:

- контроль доступу і змін. Можливість контролю доступу на фізичному рівні до елементів даних і контролю змін. Контроль доступу включає можливості визначення прав доступу до компонентів, а також вилучення елементів даних для модифікації, блокування доступу до них на час модифікації і повернення назад до репозиторію;

- відстеження модифікацій. Фіксація і ведення журналу всіх модифікацій, внесених в систему в процесі розробки або супроводу її промислової експлуатації;

- керування версіями. Ведення та контроль даних про версії системи та всіх її колективно використаних компонентах;

- облік стану об'єктів конфігураційного управління. Можливість отримання звітів про всі наявні версії, уміст і стан наявних в експлуатації об'єктів конфігураційного управління;

- генерація версій і модифікацій. Підтримка користувацького опису послідовності дій, необхідних для формування версій і модифікацій, автоматичне виконання цих дій;

- архівація. Можливість автоматичної архівації елементів даних для подальшого використання.

с. Керування проектом:

- керування роботами і ресурсами. Контроль і керування процесом проектування ІС відповідно до структури задач і призначення: виконавців, послідовності їх виконання, завершеності окремих етапів проекту і проекту в цілому. Можливість підтримки планових даних, фактичних даних та їх аналізу. Типові дані включають графіки (з урахуванням календаря, робочих годин, вихідних тощо), комп'ютерні ресурси, розподіл персоналу, бюджету тощо;

- оцінка. Можливість оцінювати витрати, графік та інші проектні параметри, що вводяться користувачами;

- керування процедурою тестування. Підтримка керування процедурами і програмою тестування, наприклад, керування розкладом планованих процедур, фіксація і запис результатів тестування, генерація звітів тощо;

- керування якістю. Введення відповідних даних, їх аналіз і генерація звітів;

- корегувальні дії. Підтримка керування корегуючими діями, включаючи обробку повідомлень про проблемні ситуації.

4.2.4.1. Надійність

До розділу надійності відносять:

- адміністрування сховища. Контроль та забезпечення цілісності проектних даних;

- автоматичне резервування (обумовлене постачальником або плановане користувачем);

- безпеку. Захист від несанкціонованого доступу;

- обробку помилок. Виявлення помилок в роботі системи, повідомлення користувача, коректне завершення роботи або збереження стану до моменту переривання;

- аналіз відмов у критичних ситуаціях.

4.2.4.2. Простота використання

Під простотою використання розуміється:

- зручність для користувача інтерфейсу. Зручність виклику, розташування та подання часто використовуваних елементів екрану, способів введення даних тощо;

- локалізація (відповідно до вимог даної країни);

- простота освоєння. Трудові і часові витрати на освоєння засобів програмного забезпечення та відповідного технічного обладнання;

- адаптованість до конкретних вимог користувача. Адаптованість до різних алфавітів, режимів текстового і графічного представлення (зліва-направо, зверху-вниз) різним форматам дати, способам введення/виведення (екранним формам і форматам), змін у методології (змін графічних нотацій, правил, властивостей і складу зумовлених об'єктів) тощо;

- якість документації (повнота, зрозумілість, зручність читання, корисність тощо);

- доступність і якість навчальних матеріалів. Вони можуть включати комп'ютерні навчальні матеріали, навчальні посібники, курси;

- вимоги до рівня знань. Кваліфікація і досвід, необхідні для ефективного використання CASE-засобів;

- простота роботи з CASE-засобом (як для початківців, так і для досвідчених користувачів);

- уніфікованість інтерфейсу (по відношенню до інших засобів, що використовується в даній організації);

- он-лайнові підказки (повнота і якість);

- якість діагностики (зрозумілість і корисність діагностичних повідомлень для користувача);

- допустимий час реакції на дії користувача (залежно від середовища);

- простота встановлення і оновлення версій.

4.2.4.3. Ефективність

До ефективності відносять:

- вимоги до технічних засобів. Вимоги до оптимального розміру зовнішньої і оперативної пам'яті, типу і продуктивності процесора, що забезпечує прийнятний рівень продуктивності;

- ефективність робочого навантаження. Ефективність виконання CASE-засобом своїх функцій залежно від інтенсивності роботи користувача (наприклад, кількість натискань клавішів або кнопки миші, необхідних для виконання певних функцій);

- продуктивність. Час, що витрачається CASE-засобом для виконання конкретних завдань (наприклад, час відповіді на запит, час аналізу 100000 рядків програмного коду тощо). У деяких випадках дані оцінки продуктивності можна отримати із зовнішніх джерел.

4.2.4.4. Супровід

Основні аспекти супроводу:

- рівень підтримки з боку постачальника (швидкість вирішення проблем поставки нових версій, забезпечення додаткових можливостей);

- трасування оновлень (простота освоєння відмінностей нових версій від існуючих);

- сумісність оновлень (сумісність нових версій з існуючими, включаючи, наприклад, сумісність з вхідними або вихідними даними);

- супроводжуваність кінцевого продукту (простота внесення змін до ПЗ та документації впродовж всього ЖЦ ОП або системи).

4.2.4.5. Переносимість (адаптивність)

Адаптивності властиві наступні параметри:

- сумісність з версіями ОС (можливість роботи в середовищі різних версій однієї і тієї ж ОС, простота модифікації CASE-засобів для роботи з новими версіями ОС);

- адаптивність даних між різними версіями CASE-засобів;

- відповідність стандартам адаптивності. Такі стандарти включають документацію, комунікації та інтерфейс, віконний інтерфейс, мови програмування, мови запитів і ін.

4.2.4.6. Загальні критерії

Наведені нижче критерії є загальними за своєю природою і не належать до сукупності показників якості, наведеної в стандарті ISO IEC 9126: 1991:

- витрати на CASE-засіб. Включають вартість придбання, встановлення, початкового супроводу та навчання. Слід враховувати ціну для всіх необхідних конфігурацій (включаючи єдину копію, кілька копій, локальну ліцензію, ліцензію для підприємства, мережеву ліцензію);

- економічний ефект від впровадження CASE-засобу (рівень продуктивності, якості і т.д.). Така оцінка може зажадати економічного аналізу;

- профіль дистриб'ютора. Загальні показники можливостей дистриб'ютора. Профіль дистриб'ютора може включати кількісний склад його організації, стаж в бізнесі, фінансове становище, список будь-яких додаткових програмних продуктів, ділові зв'язки (зокрема, з іншими дистриб'юторами даного засобу), плановану стратегію розвитку;

- сертифікація постачальника. Сертифікати, отримані від спеціалізованих організацій в галузі створення ПЗ (наприклад, SEI і ISO), які засвідчують, що кваліфікація постачальника в галузі створення і супроводу ПЗ задовольняє деяким мінімально необхідним або цілком певним вимогам. Сертифікація може бути неформальною, наприклад, на основі аналізу якості роботи постачальника;

- ліцензійна політика. Доступні можливості ліцензування, право копіювання (носіїв і документації), будь-які обмеження та/або штрафні санкції за вторинне використання (мається на увазі продаж користувачем CASE-засобів, до складу яких входять деякі

компоненти даного CASE-засобу, що використовувалися при розробці програмних продуктів);

- експортні обмеження;

- профіль продукту. Загальна інформація про програмний продукт, включаючи термін його існування, кількість проданих копій, наявність, розмір і рівень діяльності користувацької групи, система звітів про проблеми, програма розвитку продукту, сукупність застосувань, наявність помилок та ін.;

- підтримка постачальника. Доступність, реактивність і якість послуг, що надаються постачальником для користувачів CASE-засобів. Такі послуги можуть включати телефонну "гарячу лінію", місцеву технічну підтримку, підтримку в самій організації;

- доступність і якість навчання. Навчання може проводитися на території постачальника, користувача або де-небудь в іншому місці;

- адаптація, необхідна для впровадження CASE-засобів в організації користувача. Прикладом може бути визначення способу використання централізованого CASE-засобу з єдиної спільної БД в розподіленому середовищі.

4.2.5. Приклад підходу до визначення критеріїв вибору CASE-засобів

Передбачається, що CASE-засоби будуть використані в великому типовому проекті ІС, який визначається характеристиками, перерахованими у вступі. У загальному випадку стратегія вибору CASE-засобів для конкретного застосування залежить від цілей, потреб і обмежень майбутнього проекту ІС.

Слід підкреслити, що визначальним фактором при виборі тих чи інших інструментальних засобів є методологія та технологія проектування, які використовуються, а не навпаки. З цієї точки зору безглуздо порівнювати CASE-засоби самі по собі - у відриві від методології, оскільки ІС можна в принципі розробити будь-якими засобами.

Традиційно під час обговорення проблеми вибору CASE-засобів велика увага приділялася особливостям реалізації тієї чи іншої методології аналізу предметної області (ER, IDEF0, IDEF1X, Gane/Sarson, Yourdon, Barker та ін.).

Безумовно, багатство образотворчих і описових засобів дає можливість на етапах стратегічного планування та аналізу побудувати найбільш повну та адекватну модель предметної області. З іншого боку, якщо говорити про кінцеві результати - бази даних і додатки, виявляється, що частина описів в них практично не відбивається, залишаючись суто декларативною (на виході в будь-якому випадку буде отримано опис БД в табличному поданні з мінімальним набором обмежень цілісності і здійсненним кодом програм, велику частину яких складають екранні форми, що не виводяться безпосередньо з моделей предметної області).

Досвідчені аналітики і проектувальники завжди з більшими або меншими трудовитратами прийдуть до потрібного кінцевого результату незалежно від того, яка конкретно методологія або її різновид реалізовані в даному інструменті. Це, звичайно, не означає, що методологія не важлива, навпаки, відсутність або неповнота описових засобів можуть з самого початку значно ускладнити роботу над проектом. Однак, найчастіше на першому плані виявляються інші критерії, невиконання яких може породити

набагато більше конфліктних ситуацій при подальшому проектуванні.

Як було зазначено в підрозділі 1.3, технологія проектування повинна бути підтримана комплексом узгоджених CASE-засобів, що забезпечують автоматизацію процесів, які виконуються на всіх стадіях ЖЦ. Може скластися враження, що якщо можна сформувати необхідну апаратну платформу з компонентів різних фірм-виробників, то так само просто можна вибрати і скомплектувати різні інструментальні засоби, кожен з яких є одним зі світових лідерів у своєму класі. Однак для інструментальних засобів в даний час, на відміну від обладнання, відсутні міжнародні стандарти на основні властивості кінцевих продуктів (програм, баз даних і їх сполучень). Оскільки складові частини проекту повинні бути інтегровані в єдиний продукт, має сенс розглядати не будь-які, а тільки зв'язані інструментальні засоби, які в принципі можуть бути орієнтовані - навіть усередині одного класу - на різні методології. При цьому необхідно відбирати до складу комплексу CASE-засобів тільки ті, що підтримують принаймні близькі методології, якщо не одну й ту ж.

Виходячи з перерахованих вище міркувань, в якості основних критеріїв вибору CASE-засобів приймаються наступні критерії:

- Підтримка повного життєвого циклу ІС із забезпеченням еволюційності її розвитку.

- Повний життєвий цикл ІС повинен підтримуватися комплексом інструментальних засобів, перелічених у розділі 3.2, з урахуванням таких особливостей:

- колективного характеру розробки моделей, проектних специфікацій та програм територіально розподіленими групами фахівців з використанням різних інструментальних засобів (включаючи їх інтеграцію, тестування і налагодження);

- необхідності адаптації типового проекту до різних системно-технічних платформ (технічних засобів, операційних системам і СУБД) та організаційно-економічних особливостей об'єктів впровадження;

- необхідності інтеграції з існуючими розробками (включаючи реінжиніринг додатків і конвертування БД). Для існуючих ІС повинен забезпечуватися плавний перехід зі старого середовища експлуатації в нове з мінімальними переробками і підтримкою експлуатованих баз даних і програм, запроваджених до початку робіт по створенню нової системи.

Забезпечення цілісності проекту і контролю за його станом.

Дана вимога означає наявність єдиного технологічного середовища створення, супроводження та розвитку ІС, а також забезпечення цілісності сховища даних.

Єдине технологічне середовище має забезпечуватися за рахунок використання єдиного CASE-засобу для підтримки моделей ІС, а також за рахунок наявності програмно-технологічних інтерфейсів між окремими інструментальними засобами, які сертифікуються і підтримуються фірмами-розробниками відповідними засобами. Зокрема, інтерфейс між CASE-засобами і засобами розробки додатків повинен виконувати дві основні функції:

а) безпосередній перехід в рамках єдиного середовища від опису логіки програми, реалізованої CASE-засобом, до розробки інтерфейсу користувача (екранних форм);

б) перенесення опису БД з репозиторію CASE-засобу в репозиторій засобів розробки додатків і в зворотному напрямку.

Вся інформація про проект повинна автоматично поміщатися в базу проектних даних, при цьому повинні підтримуватися узгодженість, несуперечність, повнота і мінімальна надлишковість проекту, а також коректність операцій його редагування.

Це може бути досягнуто за умови виключення або суттєвого обмеження можливості актуалізації репозиторію різними засобами.

У рамках CASE-засобу повинен забезпечуватися контроль відповідності декомпозицій діаграм, а також контроль відповідності діаграм різних типів (наприклад, діаграм потоків даних і ER-діаграм).

Невиконання вимоги цілісності в умовах роз'єднаності розробників і тимчасової протяжності великого проекту може означати втрату контролю за його станом.

Незалежність від програмно-апаратної платформи і СУБД.

Вимога визначається неоднорідністю середовища функціонування ІС. Така незалежність може мати дві складові:

- незалежність середовища розробки;
- незалежність середовища експлуатації додатків.

Вони забезпечуються за рахунок наявності сумісних версій CASE-засобів для різних платформ і драйверів відповідних мережевих протоколів, менеджерів транзакцій і СУБД.

Підтримка одночасної роботи груп розробників.

Розвинені CASE-засоби повинні мати можливості поділу повноважень персоналу розробників та об'єднання окремих робіт в загальний проект. Повинна забезпечуватися одночасна (в заданій мережевій конфігурації) робота проектувальників БД і розробників додатків (розробники додатків в такій ситуації можуть починати роботу з базою даних, не чекаючи повного завершення її проектування CASE-засобами). При цьому всі групи спеціалістів повинні бути забезпечені адекватним інструментарієм, а внесення змін до проекту різними розробниками повинно бути узгодженим і коректним.

Кожен розробник повинен мати можливість роботи зі своїм особистим репозиторієм, що є фрагментом або копією загального сховища. Повинні забезпечуватися:

- змістовна інтеграція всіх змін, що вносяться розробниками загалом в репозиторії;

- одночасна доступність для розробника загального та особистого репозиторіїв;

- простота переносу об'єктів між репозиторіями.

Можливість розробки програм "клієнт-сервер" необхідної конфігурації.

Мається на увазі поєднання наявності розвиненого графічного середовища розробки додатків (багатовіконість, різноманітність стандартних графічних об'єктів, різноманітність використовуваних шрифтів і т.д.) з можливістю декомпозиції (partitioning) додатку на "клієнтську" частину, що реалізує користувацький інтерфейс і екранну, "серверну" частину.

При цьому повинна забезпечуватися можливість переміщення окремих компонентів програми між "клієнтом" і

"сервером" на найбільш підходящу платформу, що забезпечує максимальну ефективність функціонування програми в цілому, а також можливість використання сервера додатків (менеджера транзакцій).

Відкрита архітектура і можливості експорту/імпорту.

Відкрита і загальнодоступна інформація про використанні формати даних і прикладні програмні інтерфейси повинна дозволяти інтегрувати інструментальні засоби третіх фірм і відносно безболісно переходити від однієї системи до іншої.

Можливості експорту/імпорту означають, що специфікації, отримані на етапах аналізу, проектування та реалізації для однієї ІС, можуть бути використані для проектування іншої ІС.

Повторне проектування і реалізація можуть бути забезпечені за допомогою засобів експорту/імпорту специфікацій в різні CASE-засоби.

Якість технічної підтримки в країні, вартість придбання і підтримки, досвід успішного використання.

Мається на увазі наявність:

- кваліфікованих дистриб'юторів і консультантів;
- швидкість обслуговування користувачів;
- висока якість технічної підтримки програмного продукту;
- навчання програмного продукту (підтримка репозиторію);
- методології застосування програмного продукту для великих колективів розробників (наявність відомостей про практику використання системи, якість документації, укомплектованість прикладами та навчальними курсами, наявність пілотних проектів).

Витрати на навчання за новими технологіями значні, проте втрати від використання сучасних складних технологій ненавченими фахівцями можуть виявитися значно вищими.

Крім того, фірми-постачальники інструментальних засобів повинні бути дієздатними, тому що технологія вибирається не на один рік, а забезпечення своєчасної та якісної підтримки на території країни потрібне повсякчас (гаряча лінія, консультації, навчання, консалтинг), можливо, через дистриб'юторів.

Що стосується вартості, слід враховувати можливість отримання безкоштовної тимчасової ліцензії, пільгової знижки вартості ліцензії на одне робоче місце CASE-засобів, знижки, надані фірмою у разі придбання великої кількості ліцензій, необхідність придбання run-time версій для експлуатації додатків і т.д.

У той же час, вартість продукту повинна розглядатися не сама по собі, а з урахуванням її відповідності можливостям програмного продукту.

Простота освоєння і використання.

Враховуються такі характеристики:

- відповідність інструменту особливостям і потенційним можливостям колективу розробників;
- доступність для користувача інтерфейсу;
- час, необхідний для навчання;
- простота запуску в промислову експлуатацію;
- якість документації;
- обсяг ручної праці при супроводженні ІС.

Забезпечення якості проектної документації.

Ця вимога ставиться до можливостей CASE-засобів аналізувати і перевіряти опис та документацію на повноту і несуперечність, а також на відповідність прийнятим у даній методології стандартам і правилам (включаючи ГОСТ, ЕСПД). В результаті аналізу повинна формуватися інформація, яка вказує на наявні суперечності або неповноту в проектній документації. Повинна бути також забезпечена можливість створювати нові форми документів, що визначаються користувачами.

Використання загальноприйнятих, стандартних нотацій і угод.

Для того, щоб проект міг виконуватися різними колективами розробників, необхідне використання стандартних методів моделювання і стандартних нотацій, які повинні бути оформлені у вигляді нормативів до початку процесу проектування. Недотримання проектних стандартів на стадії постановки технічного завдання на розробку ОП/системи:

- ставить розробників у залежність від фірми-виробника даного засобу;

- робить надто складним формальний контроль коректності проектних рішень;

- знижує можливість залучення додаткових колективів розробників, зміни виконавців і відчуження проекту, оскільки число фахівців, знайомих з даним методом (нотацією) може бути обмеженим не тільки в межах однієї проектної організації, а навіть в межах країни.

У результаті виконаного аналізу може виявитися, що жоден доступний засіб не задовольняє в потрібній мірі всі основні критерії і не покриває всі потреби проекту. У цьому випадку може

застосовуватися набір засобів, **що** дозволяє побудувати їх на базі єдиного технологічного середовища.

4.3. Виконання пілотного проекту

Перед повномасштабним впровадженням обраного CASE-засобу в організації виконується пілотний проект, метою якого є експериментальна перевірка правильності рішень, прийнятих на попередніх етапах, і підготовка до впровадження.

Пілотний проект являє собою первісне реальне використання CASE-засобу в призначеному для цього середовищі і зазвичай має на увазі більш широкий масштаб використання CASE-засобу по відношенню до того, який був досягнутий під час оцінки.

Пілотний проект повинен володіти багатьма характеристиками реальних проектів, для яких призначений даний засіб. Він переслідує такі цілі:

- підтвердження достовірності результатів оцінки та вибору;
- визначення, чи дійсно CASE-засіб годиться для використання в даній організації, і якщо так, то визначення найбільш підходящої області його застосування;
- зібрання інформацію, необхідної для розробки плану практичного впровадження;
- набуття власного досвіду використання CASE-засобу.

Пілотний проект дозволяє отримати важливу інформацію, необхідну для оцінки якості функціонування CASE-засобу та його підтримки з боку постачальника після того, як засіб впроваджено.

Важливою функцією пілотного проекту є прийняття рішення щодо придбання або відмови від використання CASE-засобу.

Провал пілотного проекту дозволяє уникнути більш значних і дорогих невдач надалі, оскільки пілотний проект зазвичай пов'язаний з придбанням відносно невеликої кількості ліцензій та навчанням вузького кола фахівців.

Первісне використання нової CASE-технології в пілотному проекті має ретельно плануватися і контролюватися. Пілотний проект включає наступні кроки (рисунок 4.3.1).

Визначення характеристик пілотного проекту.

Пілотний проект повинен мати наступні характеристики:

- область застосування. Щоб полегшити остаточне визначення області застосування CASE-засобу, предметна область пілотного проекту повинна бути типовою для звичайної діяльності організації. Пілотний проект має допомогти визначити будь-яку додаткову технологію, окреслити сферу навчання або підтримки, які необхідні для переходу від пілотного проекту до широкомасштабного використання засобу. У рамках цих обмежень пілотний проект повинен мати невеликий, але значущий розмір;

- масштабованість. Результати, отримані в пілотному проекті, повинні показати масштабованість засобів. Мета - отримати чітке уявлення про масштаби проектів, для яких даний засіб можна застосувати;

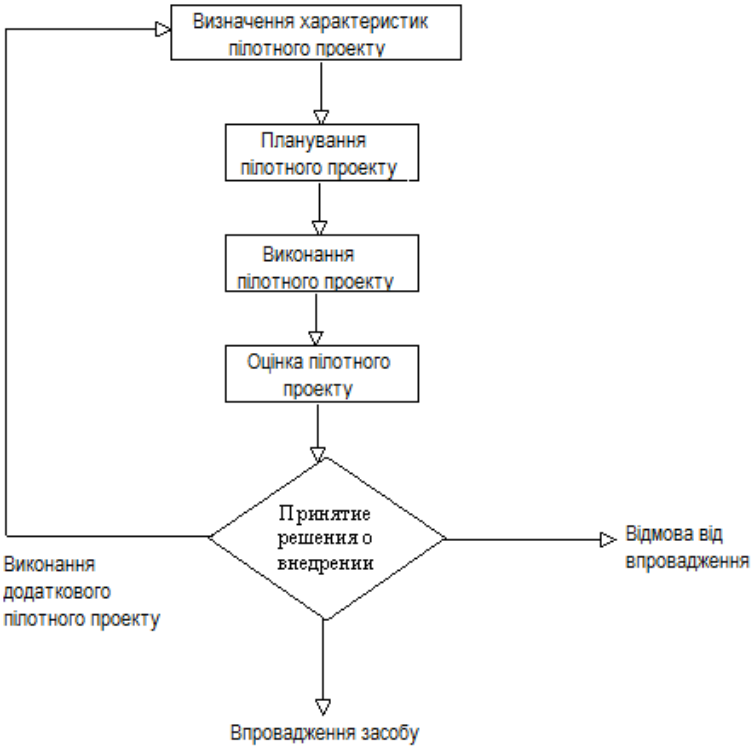

Рисунок 4.3.1 - Кроки пілотного проекту

- наочність. Пілотний проект не повинен бути незвичайним або унікальним для організації. CASE-засіб повинен використовуватися для вирішення завдань, що відносяться до предметної області і доступними для розуміння кожного проектувальника, незалежно від кваліфікації;

- критичність. Пілотний проект повинен мати істотну значущість, щоб опинитися в центрі уваги, але не повинен бути критичним для успішної діяльності організації в цілому. Необхідно усвідомлювати, що початкове впровадження нової технології-це

певний ризик. При виборі пілотного проекту доводиться вирішувати наступну дилему: успіх незначного проекту може залишитися непоміченим, з іншого боку, провал значимого проекту може викликати надмірну критику;

- авторитетність. Група фахівців, що беруть участь в проекті, повинна володіти високим авторитетом, при цьому результати проекту будуть серйозно сприйняті іншими співробітниками організації;

- характеристики проектної групи. Проектна група повинна мати готовністю до нововведень, технічну зрілість і прийнятний рівень досвіду та знань у даній технології і предметній області. З іншого боку, група повинна відображати в мініатюрі характеристики всієї організації в цілому.

У більшості випадків існує баланс між бажанням реалізувати ідеальний пілотний проект і реальними обмеженнями організації. Організація повинна вибрати пілотний проект таким чином, щоб, по-перше, спосіб використання CASE-засобу в ньому збігався з подальшими планами, і, по-друге, перераховані вище характеристики були збалансовані з реальними умовами організації.

Крім того, організація повинна враховувати тривалість пілотного проекту (і в цілому процесу впровадження). Занадто тривалий проект пов’язаний з ризиком втрати інтересу до нього з боку керівництва.

Планування пілотного проекту.

Планування пілотного проекту повинне вписуватися в звичайний процес планування проектів в організації. План повинен містити таку інформацію:

- цілі, завдання та критерії оцінки;
- персонал;
- процедури і угоди;
- навчання;
- графік і ресурси.

Цілі, завдання та критерії оцінки.

Очікувані результати пілотного проекту повинні бути чітко визначені. Ступінь відповідності з цими результатами являє собою основу для подальшої оцінки проекту. Для визначення цілей, завдань та критеріїв оцінки необхідно виконати наступні дії:

- описати проект в термінах очікуваних результатів (тобто - кінцевого продукту). Інформація повинна включати форму подання і зміст результатів. Мають бути чітко визначені договірні вимоги та відповідні стандарти;

- визначити загальні цілі проекту. Прикладом цілі може бути визначення ступені поліпшення якості проектної документації в результаті застосування CASE-засобів;

- визначити конкретні завдання, які реалізують поставлені цілі. Кожній меті можна поставити у відповідність одну або декілька конкретних задач з кількісно оцінюваними результатами. Прикладом такого завдання може бути порівняльний аналіз якості документації, отриманої за допомогою CASE-засобу і без нього. Документація може включати специфікацію вимог до ПЗ, високорівневі і детальні проектні специфікації;

- визначити критерії оцінки результатів. Щоб визначити ступінь успіху пілотного проекту, необхідно використовувати набір критеріїв, заснованих на згаданих вище завданнях. Прикладом критерію може бути ступінь несуперечливості проектної документації та контрольованості виконання вимог до ПЗ. Значення критеріїв повинні порівнюватися з базовими значеннями, отриманими до виконання пілотного проекту.

Персонал.

Фахівці, вибрані для участі в пілотному проекті, повинні мати відповідний авторитет і вплив та бути прихильниками нової технології. Група повинна включати як технічних фахівців, так і менеджерів, які мають бути зацікавленими у новій технології та які володіють методикою її використання. Група повинна володіти високими здібностями до комунікації, знанням особливостей організаційних процесів і процедур, а також предметної області. Група не має, тим не менше, складатися повністю з фахівців вищої ланки, вона повинна представляти середній рівень організації.

Багато CASE-засобів забезпечують можливості, пов’язані з генерацією проектної документації та конфігураційним керуванням. Фахівці, пов’язані з цими та іншими суміжними аспектами розробки та супроводу ПЗ, також повинні бути включені до складу групи.

Після завершення пілотного проекту група повинна бути відкрита для обміну інформацією з іншими фахівцями організації щодо можливостей нового засобу і досвіду, отриманого при його використанні. Може виявитися потреба розосередження членів проектної групи по всій організації з метою розповсюдження їх досвіду і знань.

Процедури і угоди.

Необхідно чітко визначити процедури і угоди, що регулюють використання CASE-засобів в пілотному проекті. Це завдання може виявитися більш довгим і складним, ніж очікується, при цьому може виявитися необхідним залучення сторонніх експертів.

Прикладами процедур і угод, які можуть вплинути на успіх пілотного проекту, є методологія, технічні угоди (зокрема, з найменувань і структур каталогів, стандартів проектування та програмування - див підрозділ 1.3) та організаційні угоди (зокрема, облік використання ресурсів, авторизація, контроль змін, процедури експертизи та підготовки звітів, стандарти перевірки якості).

У пілотному проекті повинні використовуватися прийняті в організації процедури і угоди. З іншого боку, протягом пілотного проекту процедури і угоди мають тенденцію до розвитку і вдосконалення у міру накопичення досвіду застосування засобу. Існуючі процедури і угоди можуть виявитися неефективними або надто обмежувальними. При цьому, ті зміни, які пропонується в них вносити, повинні документуватися.

Навчання.

Повинні бути визначені види і обсяг навчання, необхідного для виконання пілотного проекту. При плануванні навчання потрібно мати на увазі три види потреб: технічні, управлінські та мотиваційні. Ресурси, необхідні для навчання (навчальні аудиторії та обладнання, викладачі та навчальні матеріали), повинні відповідати плану пілотного проекту.

Графік навчання має визначати як фахівців, які підлягають навчанню, так і види навчання, яке вони повинні пройти. Навчання, яке проводиться в період виконання проекту, має починатися якомога швидше після початку проекту. Навчання по засобам, процесам чи методам, які не будуть використовуватися протягом декількох місяців після початку проекту, має плануватися на той час, коли в них виникне реальна потреба.

Постачальники CASE-засобів звичайно пропонують навчання по використанню з поставленими їм засобам. Крім цього, для деяких засобів може бути необхідно навчання методології. Деякі види навчання повинні виконуватися власними силами. Такі види навчання включають використання CASE-засобу в контексті процесів, що відбуваються в організації, а також у сукупності з іншими засобами в даному середовищі. Частина плану пілотного проекту, пов'язана з навчанням, повинна використовуватися в якості входу для плану практичного впровадження.

При виборі необхідного навчання повинні братися до уваги такі фактори:

- кваліфікація викладачів;

- відповідність навчання характеристикам конкретних груп спеціалістів (наприклад, оглядові курси для менеджерів, докладні курси для розробників);

- можливість проведення курсів безпосередньо на робочих місцях;

- можливість проведення розширених курсів;

- можливість підготовки власних викладачів.

Графік та ресурси.

Повинен бути розроблений графік, що включає ресурси та терміни (етапи) проведення робіт. Ресурси включають персонал, технічні засоби, програмне забезпечення та фінансування. Дані про персонал можуть визначати конкретних фахівців або вимоги до кваліфікації, необхідної для успішного виконання пілотного проекту. Фінансування має визначатися окремо по кожному виду роботи: придбання CASE-засобів, встановлення, навчання, окремі етапи проектування.

Виконання пілотного проекту.

Пілотний проект має виконуватися згідно плану. Організаційна діяльність, пов'язана з виконанням пілотного проекту та підготовкою звітів, повинна виконуватися у встановленому порядку. Пілотна природа проекту вимагає спеціальної уваги до питань придбання, підтримки, експертизи та оновлення версій. Ці питання розглядаються нижче.

Придбання, встановлення та інтеграція.

Після того, як CASE-засіб вибрано, він повинен бути придбаний, інтегрований в проектне середовище і налаштований відповідно до вимог пілотного проекту. Межі цієї діяльності залежать від тих дій, які мали місце в процесі оцінки і вибору, а також від ступені модифікації засобів, необхідних для їх використання в проекті.

Процес придбання може включати підготовку контракту, переговори, ліцензування та іншу діяльність, яка виходить за рамки даних рекомендацій. Ця діяльність потребує витрат часу і людських ресурсів, які повинні бути враховані при плануванні. План повинен передбачати можливість відмови від обраного засобу на даному етапі через договірні розбіжності.

Після того, як процес придбання завершено, засіб має бути встановлено, відтестовано і прийнято в експлуатацію. Тестування дозволяє переконатися, що поставлений продукт відповідає вимогам контракту, володіє необхідною повнотою і коректністю. Етап приймання може бути передбачений контрактом, його реальний термін може відрізнятися від того, який був передбачений спочатку в плані пілотного проекту. Особливу увагу необхідно приділити дотриманню всіх вимог постачальника до параметрів середовища функціонування CASE-засобу.

Після завершення придбання може знадобитися налаштування та інтеграція. Налаштування включає модифікацію інтерфейсів, пов'язану з вимогами фахівців проектної групи, а також встановлення прав доступу і привілеїв. Налаштування повинне залишатися в рамках тих можливостей, які надає сам засіб. Не слід займатися модифікацією готових продуктів на рівні вихідних кодів.

Якщо новий засіб має використовуватися в сукупності з деякими іншими засобами, необхідно визначити взаємодію засобів та потрібну інтеграцію. Для інтеграції нових засобів з існуючими може знадобитися побудова спеціальних оболонок. Складна інтеграція може зажадати залучення сторонніх експертів.

Підтримка.

Доступна підтримка повинна включати (за згодою) "гарячу лінію" постачальника і підтримку місцевого постачальника, підтримку в самій організації, контакти з досвідченими користувачами в інших організаціях та участь у роботі груп користувачів.

Внутрішня підтримка повинна здійснюватися фахівцями, знайомими зі встановленням засобів та роботою з ними. Існує декілька можливих варіантів отримання такої підтримки (наприклад, від фахівця даної організації, що має досвід попередньої роботи із засобом; учасників процесу оцінки та вибору або досвідченого консультанта). Такий тип підтримки повинен спеціальним чином плануватися і адмініструватися. Особлива увага повинна бути приділена засобам, які працюють в мережах або володіють репозиторіями, що підтримують багатокористувацьку роботу.

Періодичні експертизи.

Звичайні процедури експертизи проектів, що існують в організації, повинні виконуватися і для пілотного проекту, особлива увага повинна приділятися саме пілотним аспектам проекту. Крім цього, результати експертиз повинні служити мірою успішного використання CASE-засобів.

Оновлення версій.

Користувачі CASE-засобів можуть очікувати періодичного оновлення версій з боку постачальника протягом виконання пілотного проекту. При цьому необхідно ретельне ставлення до інтеграції цих версій. Слід заздалегідь оцінити вплив цих оновлень на хід проекту. Нові версії можуть як забезпечити нові можливості,

так і породити нові проблеми. Наприклад, нова версія може зажадати видозміненого або додаткового навчання, а також може справити негативний вплив на вже виконану до цього моменту роботу.

Оцінка пілотного проекту.

Після завершення пілотного проекту його результати необхідно оцінити і зіставити їх з початковими потребами організації, критеріями успішного впровадження CASE-засобів, базовими метриками і критеріями успіху пілотного проекту. Така оцінка повинна встановити можливі проблеми і найважливіші характеристики пілотного проекту, які можуть вплинути на придатність CASE-засобів для організації. Вона повинна також вказати проекти або структурні підрозділи всередині організації, для яких даний засіб є підходящим. Крім цього, оцінка може дати інформацію щодо вдосконалення процесу впровадження в подальшому.

У процесі оцінки пілотного проекту організація повинна визначити свою позицію з наступних трьох питань:

- Чи доцільно впроваджувати CASE-засіб?

- Які конкретні особливості пілотного проекту призвели до його успіху (або невдачі)?

- Які проекти або підрозділи в організації могли б отримати вигоду від використання засобів?

Прийняття рішення про доцільність впровадження CASE-засобів.

На даному етапі процесу впровадження організація повинна зробити суттєві інвестиції в CASE-засоби. Якщо CASE-засоби задовольнили або навіть перевищили очікування організації, то рішення про впровадження може бути достатньо простим і швидким.

З іншого боку, може виявитися, що в рамках пілотного проекту засоби не виправдали тих очікувань, які на них покладалися, або ж у пілотному проекті вони використовувалися задовільно, однак досвід показав, що подальші вкладення в засоби не гарантують успіху.

Можливі чотири категорії результатів і відповідних дій:

- пілотний проект зазнав невдачі, і його аналіз показав неадекватність очікувань організації. У цьому випадку організація може переглянути результати проекту в контексті більш реалістичних очікувань;

- пілотний проект зазнав невдачі, і його аналіз показав, що обрані засоби не задовольняють потреби організації. У цьому випадку організація може ухвалити рішення не запроваджувати ці засоби, однак при цьому також вона має переглянути свої потреби і підхід до оцінки та вибору CASE-засобів;

- пілотний проект зазнав невдачі і його аналіз показав наявність таких проблем, як невдалий вибір пілотного проекту, неадекватне навчання і брак ресурсів. У цьому випадку може виявитися досить складно прийняти рішення про те, чи слід знову виконати пілотний проект, продовжити роботу по впровадженню або відмовитися від CASE-засобів. Однак, незалежно від

прийнятого рішення, процес впровадження потребує перегляду і підвищеної уваги;

- пілотний проект завершився успішно, і визнано за доцільне впроваджувати CASE-засоби в деяких підрозділах або, можливо, у всій організації в цілому. В цьому випадку наступним кроком є визначення найбільш відповідного масштабу впровадження.

У ряді випадків аналіз пілотного проекту може показати, що причиною невдачі став більш ніж один фактор. Наступні спроби впровадження CASE-технології повинні чітко виявити всі причини невдачі. В екстремальних випадках ретельний аналіз може показати, що в даний момент організація просто не готова до успішного впровадження складних CASE-засобів. У такій ситуації організація може спробувати вирішити свої проблеми іншими засобами.

Особливості пілотного проекту.

Дуже важливо провести аналіз пілотного проекту з тим, щоб визначити його елементи, які є критичними для успіху і визначити ступінь відображення цими елементами діяльності організації в цілому. Наприклад, якщо в пілотному проекті беруть участь найкращі програмісти організації, він може закінчитися успішно навіть всупереч використанню CASE-засобів, а не завдяки їм. З іншого боку, CASE-засоби можуть бути застосовані для розробки програми, до якої вони явно не підходять за своїми характеристиками. Тим не менш, таке використання могло б вказати на область найбільш раціонального застосування засобів в даній організації.

Найважливіші характеристики пілотного проекту, які не є представницькими для організації в цілому, можуть включати наступні аспекти:

- процеси в пілотному проекті в чому-небудь відрізняються від процесів у всій організації;

- кваліфікація групи спеціалістів пілотного проекту не відображає кваліфікацію інших фахівців організації;

- ресурси, виділені на виконання проекту, значно перевищують ті, які виділяються для звичайних проектів;

- предметна область або масштаб проекту не відповідають іншим проектам.

Переваги використання CASE-засобів.

Пілотний проект варто зіставити з діяльністю організації в цілому з тим, щоб визначити найбільш істотну подібність і відмінність. Наприклад, якщо найбільш зацікавлені й кваліфіковані учасники проекту зіткнулися з серйозними труднощами в освоєнні засобів, то менш зацікавленим і кваліфікованим програмістам з інших підрозділів потрібний істотно більший термін навчання.

Пілотний проект може показати, що засоби доцільно використовувати для деяких класів проектів і недоцільно для інших. Наприклад, засіб формальної верифікації може підходити для життєво важливих програм і не підходити для менш критичних додатків.

Перед розробкою плану переходу організація повинна оцінити очікуваний ефект для різних підрозділів або класів проектів. При цьому слід враховувати, що деякі підрозділи можуть не володіти необхідною кваліфікацією або ресурсами для використання CASE-засобів.

Прийняття рішення про впровадження.

Можливим рішенням має бути одне з наступних:

- впровадити засіб. У цьому випадку рекомендується масштаб впровадження, який повинен бути визначений в термінах структурних підрозділів та предметної області;

- виконати додатковий пілотний проект. Такий варіант має розглядатися тільки в тому випадку, якщо залишилися конкретні невирішені питання щодо впровадження CASE-засобу в організації. Новий пілотний проект повинен бути таким, щоб відповісти на ці питання;

- відмовитись від засобу. У цьому випадку причини відмови від конкретного засобу повинні бути визначені в термінах потреб організації або наявністю критеріїв, які залишилися незадоволеними. Перед тим, як продовжити діяльність по впровадженню CASE-засобів, потреби організації повинні бути переглянуті на предмет своєї обґрунтованості;

- відмовитись від використання CASE-засобів взагалі. Пілотний проект може показати, що організація або не готова до впровадження CASE-засобів, або автоматизація даного аспекту процесу створення і супроводу ПЗ не дає ніякого ефекту для організації. У цьому випадку причини відмови від CASE-засобів повинні бути також визначені в термінах потреб організації або критеріїв, які залишилися незадоволеними. При цьому необхідно розуміти відмінність цього варіанту від попереднього, пов’язаного з вадами конкретного засобу.

Результатом даного етапу є документ, в якому обговорюються результати пілотного проекту і деталізуються рішення з впровадження.

4.4. Перехід до практичного використання CASE-засобів

Процес переходу до практичного використання CASE-засобів починається з розробки та подальшої реалізації плану переходу. Цей план може відображати поетапний підхід до переходу, починаючи з ретельно обраного пілотного проекту до проектів з істотно збільшеною різноманітністю характеристик.

Розробка плану переходу.

План переходу повинен включати наступне:

- інформацію щодо цілей, критеріїв оцінки, графіки і можливих ризиків, пов'язаних з реалізацією плану;
- інформацію щодо придбання, встановлення та налаштування CASE-засобів;
- інформацію щодо інтеграції кожного засобу з існуючими засобами, включаючи як інтеграцію CASE-засобів один з одним, так і їх інтеграцію в процеси розробки та експлуатації ПЗ, що існують в організації;
- очікувані потреби в навчанні та ресурси, що використовуються протягом і після завершення процесу переходу від одної технології програмування до іншої;
- визначення стандартних процедур використання CASE-засобів.

Цілі, критерії оцінки, графік і ризики, пов'язані з планом переходу.

Дана інформація повинна включати наступне:

- типи проектів, в яких в кінцевому рахунку буде використовуватися CASE-засіб;

- графік переходу до практичного використання CASE-засобів в окремих проектах, які включають необхідну підготовку до його широкого використання;

- графік впровадження CASE-засобів з урахуванням навчання користувачів, які будуть ці засоби експлуатувати;

- можливі ризики і непередбачені обставини;

- джерела існуючих (базових) даних і метрики для оцінки змін, викликаних використанням CASE-засобів.

На додаток до сказаного, слід приділити особливу увагу питанням контролю змін. Ролі вищого керівництва, суб'єктів та об'єктів змін повинні бути уточнені в порівнянні з пілотним проектом, оскільки технологія підлягає широкому розповсюдженню в організації.

Мається на увазі, що план переходу успішно виконаний, коли не потрібно більше спеціального планування підтримки використання засобів. У цей момент використання засобів узгоджується з обсягом очікуваних проектних процедур та трудомісткістю їх виконання. План роботи з ними включається в загальний план поточної підтримки ПЗ, що існує в організації.

Придбання, встановлення і налаштування засобів.

Придбання, встановлення і налаштування, виконані в рамках пілотного проекту, можуть знадобитися в більш широкому аспекті. При цьому необхідна наступна інформація:

- сукупність програмних компонентів, документації і навчання, які необхідно придбати для кожної окремої платформи;

- механізм отримання нових версій;

- налагодження засобів, необхідних для виконання існуючих в організації процедур і угод;

- особа або підрозділ, відповідальний за встановлення, інтеграцію, налаштування та експлуатацію засобів;

- план конвертації даних та зняття старих засобів з експлуатації.

Завдання придбання, встановлення та налаштування повинні бути якомога швидше передані з групи пілотного проекту в існуючу службу системної підтримки ПЗ організації.

Інтеграція засобів з існуючими засобами і процесами.

Інтеграція нового засобу з існуючими засобами і процесами є важливим кроком у повномасштабному впровадженні засобів. У більшості випадків така інтеграція в процесі пілотного проектування не здійснюється, проте накопичена в цьому процесі інформація може допомогти в розробці планів інтеграції. Для планування інтеграції необхідна наступна інформація:

- найменування і версії існуючих CASE-засобів, за якими має інтегруватися новий засіб;

- опис даних, які повинні спільно використовуватися новим ПЗ та існуючими програмними засобами, а також попередня інформація про джерела цих даних;

- опис інших взаємозв'язків між новим та існуючими засобами (таких, як зв'язок з передачі керування до порядку використання), а також попередня інформація про механізми підтримки цих взаємозв'язків;

- оцінки витрат, термінів і ризиків, пов'язаних з інтеграцією (і, можливо, переходом від існуючих засобів і даних);

- визначення способу впровадження даного засобу в діяльність по вдосконаленню існуючих процесів;

- очікувані зміни в існуючих процесах і продуктах, які є наслідком використання нового засобу і оцінювані, по можливості, кількісно.

Ризик, пов'язаний з інтеграцією нового засобу з існуючими засобами і процесами, знижується, якщо потреби в інтеграції враховуються в процесі оцінки і вибору засобу.

Навчання та ресурси, що використовуються протягом і після завершення процесу переходу.

Дана інформація повинна включати наступне:

- персонал (включаючи користувачів, адміністраторів і інтеграторів), що потребує в навчанні використання CASE-засобів;

- вид навчання, необхідний для кожної категорії користувачів і обслуговуючого персоналу, з урахуванням особливої важливості навчання спільному використанню різних CASE-засобів, а також методів і процесів, пов'язаних з ними;

- вид навчання, необхідний для різних фахівців (наприклад, для групи тестування і незалежної служби сертифікації);

- частота та обсяг навчання;

- види і доступність підтримки програмного продукту та технічних засобів в процесі промислової експлуатації.

Визначення стандартів і процедур використання засобів.

План переходу повинен визначати такі стандарти та процедури використання програмних та технічних засобів:

- інструкції з моделювання та проектування;
- процедури контролю якості та процесів приймання, включаючи графік експертиз і перелік використовуваних методологій;
- процедури резервного копіювання, захисту майстер-копій і конфігурації баз даних;
- процедури інтеграції з існуючими засобами і базами даних;
- процедури спільного використання даних і контролю цілісності БД;
- стандарти та процедури забезпечення секретності;
- стандарти документування.

Стандарти використання CASE-засобів, розроблені під час пілотного проекту, повинні використовуватися в якості відправної точки для розробки більш повного набору стандартів використання засобів в даній організації (див. підрозділ 1.3). При цьому повинен враховуватися досвід учасників пілотного проекту.

Реалізація плану переходу.

Реалізація плану переходу вимагає:

- постійного моніторингу використання CASE-засобів;
- забезпечення поточної підтримки;
- забезпечення супроводу;
- забезпечення засобами в міру необхідності.

Періодичні експертизи.

Досягнуті результати повинні періодично піддаватися експертизі відповідно до графіка, план переходу повинен

коригуватися при необхідності. Постійна увага повинна приділятися ступені задоволення потреб організації і критеріїв успішного впровадження CASE-засобів.

Періодичні експертизи повинні продовжуватися і після завершення процесу впровадження. Такі експертизи можуть аналізувати метрики та іншу інформацію, що отримується у процесі роботи з CASE-засобами, щоб визначати, наскільки добре вони продовжують виконувати необхідні функції. Такі експертизи можуть також вказати на необхідність додаткової модифікації процесів.

Поточна підтримка.

Поточна підтримка необхідна для наступного:

- відповідей на питання, пов'язаних з використанням CASE-засобів;

- передачі інформації про досягнуті успіхи та отримані уроки фахівцям інших організації;

- модифікації та вдосконалення стандартів, угод та процедур, пов'язаних з використанням CASE-засобів;

- інтеграції нових CASE-засобів з існуючими та супроводження інтегрованих засобів у міру появи нових версій;

- допомоги новим співробітникам у освоєнні CASE-засобів і пов'язаних з ними процедур;

- планування та контролю оновлення версій;

- планування впровадження нових можливостей засобів в процеси проектування.

Дії, що виконуються в процесі переходу.

Для підтримки процесу переходу до практичного використання CASE-засобів бажано виконання наступних дій:

- підтримка поточного навчання. Потреба в навчанні може виникати періодично внаслідок появи нових версій CASE-засобів або залучення до проекту нових співробітників;

- підтримка рольових функцій, пов'язаних з процесом впровадження. Оскільки впровадження CASE-засобів призводить до змін у культурі проектування організації, необхідно в процесі впровадження виділити ряд ключових ролей (таких, як керівництво вищого рівня, проектної групи і цільової групи);

- методики керування оновленням версій. Ці методики можуть бути пов'язані з оновленням версій, процедурами встановлення, процедурами контролю якості для оцінки нових версій, процедурами оновлення баз даних, конфігурацією версій і середовищем підтримки (інші CASE-засоби, операційна система тощо);

- вільний доступ до інформації. Повинні бути визначені механізми, що забезпечують вільний доступ до інформації про досвід впровадження і одержаних з цього уроках, включаючи дошки оголошень, інформаційні бюлетені, призначені для користувача групи, семінари та публікації;

- налагодження тісної взаємодії робочого колективу з розробником програмного продукту. Така взаємодія дозволяє організації бути в курсі планів розробника і забезпечувати оперативне задоволення своїх вимог.

Для успішного впровадження CASE-засобів в організації істотно важливою є послідовність у їх застосуванні. Оскільки більшість систем проектуються колективно, необхідно визначити характер майбутнього використання CASE-засобів як окремими розробниками, так і групами. Використання стандартних процедур

дозволить забезпечити плавний перехід між окремими стадіями ЖЦ ПЗ.

Як правило, всі розуміють, що навчання є центральною ланкою, яка забезпечує нормальне використання CASE-засобів в організації. Проте досить поширена помилка полягає в тому, що проводиться початкове навчання для групи непідготовлених користувачів, а потім все обмежується мінімальним поточним навчанням.

Учасники пілотного проекту, що отримали початкове навчання, можуть бути високо кваліфікованими ентузіастами нової технології, що прагнуть використовувати її будь-де. З іншого боку, для розробників, які братимуть участь у проекті в подальшому, може знадобитися більш інтенсивне і глибоке навчання, а також постійна підтримка у використанні засобів.

На додаток до цього слід зазначити, що кожна категорія персоналу (наприклад, адміністратори CASE-засобів, служба підтримки робочих місць, інтегратори засобів, служба супроводження та розробники додатків) потребує різного навчання.

Навчання не повинно замикатися тільки на користувачах CASE-засобів, навчатися повинні також ті співробітники організації, на діяльність яких так чи інакше впливає використання CASE-засобів.

В процесі подальшого використання CASE-засобів в організації навчання має стати частиною процесу орієнтації принаймні нових співробітників і залучення співробітників до проектів, в яких використовуються CASE-засоби. Навчання має стати невід'ємною складовою частиною нормативних матеріалів, які пропонуються новим співробітникам.

Однак загальна помилка, яка робиться в процесі переходу, полягає в недооцінці ресурсів, необхідних для підтримки постійного використання складних CASE-засобів. Зростання необхідних ресурсів викликається трьома причинами:

- складністю CASE-засобів;
- частотою появи нових версій;
- взаємодією між засобами і зовнішнім середовищем.

Складність CASE-засобів призводить до зростання потреб у ретельному та продуманому навчанні. Крім того, багато CASE-засобів можуть використовуватися тільки кваліфікованими фахівцями, які вміють супроводжувати проектні бази даних і оперативно реагувати на виникаючі проблеми.

Висока частота оновлення версій засобів може привести до виникнення нетривіальних проблем, які часто не беруться до уваги. Такі оновлення згубно позначаються на жорстких планах і графіках роботи. Взаємодія між засобами і зовнішнім, по відношенню до них, середовищем також може іноді породжувати певні проблеми. Мається на увазі той факт, що хоча багато засобів досягли рівня мінімальної несумісності даних між окремими версіями, проблеми забезпечення сумісності з іншими елементами зовнішнього середовища залишаються в силі.

Оцінка результатів переходу.

Програма постійної оцінки якості та продуктивності ПЗ упродовж ЖЦ має важливе значення для наступного:

- визначення ступені вдосконалення процесів;
- випередження можливих стратегічних прорахунків;
- своєчасної відмови від використання застарілої технології.

Щоб визначити, наскільки ефективно новий CASE-засіб підвищує продуктивність і/або якість, організація повинна спиратися на деякі базові дані. Нажаль, лише деякі організації в даний час накопичують дані для реалізації програми поточної кількісної оцінки та удосконалення процесів. Для доказу ефективності CASE-засобів і їх можливостей покращувати продуктивність, необхідні такі базові метричні дані, як: використаний час; час, виділений персонально для конкретних фахівців; розмір, складність та якість ПЗ; зручність супроводу.

Метрична оцінка повинна починатися з реальної оцінки поточного стану середовища ще до початку впровадження CASE-засобів і підтримувати процедури постійного накопичення даних. Період часу, протягом якого виконується кількісна оцінка впливу, що чиниться впровадженням CASE-засобів, є досить значущою величиною з точки зору визначення ступені успішності переходу.

Деякі організації, що успішно впровадили в кінцевому рахунку CASE-засоби, зіткнулися з короткочасними негативними ефектами на початку процесу. Інші, успішно почавши, недооцінили довготривалі витрати на супроводження і навчання. Внаслідок цього, найбільш прийнятний часовий інтервал для оцінки ступені успішності впровадження, повинен бути достатньо великим, щоб подолати будь-які негативні ефекти на початковому етапі, а також змоделювати майбутні довготривалі витрати. З іншого боку, даний інтервал повинен відповідати цілям організації і очікуваним результатам.

Зрештою, досвід, отриманий при впровадженні CASE-засобів, може частково змінити цілі організації і очікування, які покладаються на CASE-засоби. Наприклад, організація може

зробити висновок, що CASE-засоби доцільно використовувати для більшого чи меншого кола користувачів і процесів у циклі створення та супроводу ПЗ. Такі зміни в очікуваннях часто можуть дати позитивні результати, але можуть також призвести до внесення відповідних корегувань у визначення ступені успішного впровадження CASE-засобів в даній організації.

Результатом даного етапу є впровадження CASE-засобів в повсякденну практику організації, при цьому більше не потрібно будь-якого спеціального планування. Крім того, підтримка CASE-засобів включається до плану поточної підтримки ПЗ в даній організації.

5. Характеристики CASE-засобів

5.1. Silverrun + JAM

5.1.1. Silverrun

CASE-засіб Silverrun американської фірми Computer Systems Advisers, Inc. (CSA) використовується для аналізу і проектування ІС бізнес-класу [2] і орієнтований більшою мірою на спіральну модель ЖЦ. Він застосовується для підтримки будь-якої методології, заснованої на роздільній побудові функціональної та інформаційної моделей (діаграм потоків даних і діаграм "сутність-зв'язок").

Налаштування на конкретну методологію забезпечується вибором необхідної графічної нотації моделей і набору правил перевірки проектних специфікацій. У системі є готові налаштування для найбільш поширених методологій: DATARUN (основна методологія, що підтримується Silverrun), Gane/Sarson, Yourdon/DeMarco, Merise, Ward/Mellor, Information Engineering. Для кожного поняття, введеного в проекті, є можливість додавання власних описувачів. Архітектура Silverrun дозволяє нарощувати середовище розробки в міру необхідності.

Структура та функції.

Silverrun має модульну структуру і складається з чотирьох модулів, кожен з яких є самостійним продуктом і може купуватися та використовуватися без зв'язку з іншими модулями.

Модуль побудови моделей бізнес-процесів у формі діаграм потоків даних (BPM - Business Process Modeler) дозволяє моделювати функціонування обстежуваної організації або

створюваної ІС. У модулі BPM забезпечена можливість роботи з моделями великої складності:

- автоматична перенумерація,

- робота з деревом процесів (включаючи візуальне перетягування гілок),

- від'єднання і приєднання частин моделі для колективної розробки. Діаграми можуть зображатися в декількох наступних нотаціях, включаючи Yourdon/DeMarco і Gane/Sarson. Є також можливість створювати власні нотації, у тому числі, додавати до числа зображених на схемі дескрипторів заявлені користувачем поля.

Модуль концептуального моделювання даних (ERX - Entity-Relationship eXpert) забезпечує побудову моделей даних "сутність-зв'язок", не прив'язаних до конкретної реалізації. Цей модуль має вбудовану експертну систему, що дозволяє створити коректну нормалізовану модель даних за допомогою відповідей на змістовні питання про взаємозв'язок даних.

Можлива автоматична побудова моделі даних з описами структур даних. Аналіз функціональних залежностей атрибутів дає можливість перевірити відповідність моделі вимогам третьої нормальної форми і забезпечити їх виконання. Перевірена модель передається в модуль RDM.

Модуль реляційного моделювання (RDM - Relational Data Modeler) дозволяє створювати деталізовані моделі "сутність-зв'язок", призначені для реалізації в реляційних базах даних. У цьому модулі документуються всі конструкції, пов'язані з побудовою баз даних: індекси, тригери, збереження процедури і т.д.

Гнучка змінювана нотація і розширюваність сховища дозволяють працювати з будь-якою методологією. Можливість створювати підсхеми відповідає підходу ANSI SPARC до подання схеми бази даних. Мовою підсхем моделюються як вузли розподіленої обробки, так і призначені для користувача уявлення. Цей модуль забезпечує проектування і повне документування реляційних баз даних.

Менеджер репозиторію робочої групи (WRM - Workgroup Repository Manager) застосовується як словник даних для зберігання спільної для всіх моделей інформації, а також забезпечує інтеграцію модулів Silverrun в єдине середовище проектування.

Платою за високу гнучкість і різноманітність образотворчих засобів побудови моделей є такий недолік Silverrun, як відсутність жорсткого взаємного контролю між компонентами різних моделей (наприклад, можливості автоматичного розповсюдження змін між DFD різних рівнів декомпозиції). Слід, однак, відзначити, що цей недолік може мати істотне значення тільки у випадку використання каскадної моделі ЖЦ ПЗ.

Взаємодія з іншими засобами.

Для автоматичної генерації схем баз даних у Silverrun існують мости до найбільш поширених СУБД: Oracle, Informix, DB2, Ingres, Progress, SQL Server, SQLBase, Sybase. Для передачі даних в засоби розробки додатків є мости до мов 4GL: JAM, PowerBuilder, SQL Windows, Uniface, NewEra, Delphi. Всі мости дозволяють завантажити в Silverrun RDM інформацію з каталогів відповідних СУБД або мов 4GL. Це дозволяє документувати,

перепроектувати або переносити на нові платформи вже перебуваючі в експлуатації бази даних і прикладні системи.

При використанні мосту Silverrun розширює свій внутрішній репозиторій специфічними для цільової системи атрибутами. Після визначення показників цих атрибутів генератор додатків переносить їх у внутрішній каталог середовища розробки або використовує при генерації коду мовою SQL. Таким чином, можна повністю визначити ядро бази даних з використанням всіх можливостей конкретної СУБД: тригерів, збережених процедур, обмежень цілісності БД.

При створенні програми на мові 4GL дані, перенесені з репозиторію Silverrun, використовуються або для автоматичної генерації інтерфейсних об'єктів, або для швидкого їх створення вручну.

Для обміну даними з іншими засобами автоматизації проектування, створення спеціалізованих процедур аналізу та перевірки проектних специфікацій, складання спеціалізованих звітів у відповідності з різними стандартами в системі Silverrun є три способи видачі проектної інформації в зовнішні файли:

- система звітів. Можна, визначивши вміст звіту по репозиторію, видати звіт в текстовий файл. Цей файл можна потім завантажити в текстовий редактор або включити в інший звіт;

- система експорту/імпорту. Для більш повного контролю над структурою файлів в системі експорту/імпорту є можливість визначати не тільки вміст експортного файлу, а й роздільники записів, полів в записах, маркери початку і кінця текстових полів. Файли із зазначеною структурою можна не тільки формувати, але й завантажувати в репозиторій. Це дає можливість обмінюватися

даними з різними системами: іншими CASE-засобами, СУБД, текстовими редакторами та електронними таблицями;

- зберігання репозиторію в зовнішніх файлах через ODBC-драйвери. Для доступу до даних сховища з найбільш поширених систем управління базами даних, забезпечена можливість зберігати всю проектну інформацію безпосередньо у форматі цих СУБД.

Групова робота.

Групова робота підтримується в системі Silverrun двома способами:

- у стандартній однокористувацькій версії є механізм контрольованого поділу і злиття моделей. Розділивши модель на частини, можна роздати їх декільком розробникам. Після детального доопрацювання моделі об'єднуються в єдині специфікації;

- мережева версія Silverrun дозволяє здійснювати одночасну групову роботу з моделями, що зберігаються в мережевому репозиторії на базі СУБД Oracle, Sybase або Informix. При цьому кілька розробників можуть працювати з однією і тією ж моделлю, оскільки блокування об'єктів відбувається на рівні окремих елементів моделі.

Середовище функціонування.

Існують реалізації Silverrun трьох платформ - MS Windows, Macintosh і OS/2 Presentation Manager - з можливістю обміну проектними даними між ними.

Для функціонування в середовищі Windows необхідно мати комп'ютер з процесором моделі не нижче i486 і оперативну пам'ять обсягом не менше 8 Мб (рекомендується 16 Мб). На диску повна інсталяція Silverrun займає 20 Мб.

5.1.2. JAM

Засіб розробки додатків JAM [8] (JYACC's Application Manager) - продукт фірми JYACC (США). В даний час поставляється версія JAM 7 і готується до виходу JAM 8.

Основною рисою JAM є його відповідність методології RAD, оскільки він дозволяє досить швидко реалізувати цикл розробки програми, що полягає у формуванні чергової версії її прототипу з урахуванням вимог, виявлених на попередньому кроці, і пред'явити розробку користувачу.

Структура та функції.

JAM має модульну структуру і складається з наступних компонентів:

- ядро системи;

- JAM/DBi - спеціалізовані модулі інтерфейсу до СУБД (JAM/DBi-Oracle, JAM/DBi-Informix, JAM/DBi-ODBC і т.д.);

- JAM/RW - модуль генератора звітів;

- JAM/CASEi - спеціалізовані модулі інтерфейсу до CASE-засобів (JAM/CASE-TeamWork, JAM/CASE-Innovator і т.д.);

- JAM/TPi - спеціалізовані модулі інтерфейсу до менеджерів транзакцій (наприклад, JAM/TPi-Server TUXEDO і т.д.);

- Jterm - спеціалізований емулятор X-терміналу.

Ядро системи (власне, сам JAM) є закінченим продуктом і може самостійно використовуватися для розробки додатків. Всі інші модулі є додатковими і самостійно використовуватися не можуть.

Ядро системи включає в себе такі основні компоненти:

- редактор екранів. До складу редактора екранів входять: середовище розробки екранів, візуальний репозиторій об'єктів, власна СУБД JAM - JDB, менеджер транзакцій, тести для налагодження, редактор стилів;

- редактор меню;

- набір допоміжних утиліт;

- засоби виготовлення промислової версії програми.

При використанні JAM, розробка зовнішнього інтерфейсу програми є візуальним проектуванням і зводиться до створення екранних форм шляхом розміщення на них інтерфейсних конструкцій та визначенню екранних полів введення/виведення інформації.

Проектування інтерфейсу в JAM здійснюється за допомогою редактора екранів. Програми, розроблені в JAM, мають багатовіконний інтерфейс. Розробка окремого екрана полягає в розміщенні на ньому інтерфейсних елементів, можливому (але не обов'язковому) їх угрупованні та конкретизації різних його властивостей, що включають візуальні характеристики (позиція, розмір, колір, шрифт), поведінкові характеристики (різноманітні фільтри, формати, захист від введення) і ряд властивостей, орієнтованих на роботу з БД.

Редактор меню дозволяє розробляти і налагоджувати системи меню. Реалізована можливість побудови піктографічного меню (так звані toolbar). Призначення кожного конкретного меню того чи іншого об'єкту програми здійснюється в редакторі екранів.

У ядро JAM вбудована однокористувацька реляційна СУБД JDB. Основним призначенням JDB є прототипування додатків у тих

випадках, коли робота зі штатною СУБД неможлива або недоцільна.

У JDB реалізований необхідний мінімум можливостей реляційних СУБД за винятком індексів, збережених процедур, тригерів і уявлень (view). За допомогою JDB можна побудувати БД, ідентичну цільової БД (з точністю до відсутніх в JDB можливостей) і розробити значну частину програми.

Налагоджувальник дозволяє проводити комплексне налагодження розробки. Здійснюється трасування всіх подій, що виникають в процесі виконання програми.

Утиліти JAM включають три групи:

- конвертори файлів екранів JAM в текстові. JAM зберігає екрани у вигляді двійкових файлів власного формату. У ряді випадків (наприклад для виготовлення програмної документації проекту) необхідно текстовий опис екранів;

- конфігурування пристроїв введення/виведення. JAM і додатки, побудовані з його допомогою, не працюють безпосередньо з пристроями введення/виведення. Замість цього, JAM звертається до логічних пристроїв введення/виведення (клавіатура, термінал, звіт). Відображення логічних пристроїв у фізичні здійснюється за допомогою засобів конфігурування;

- обслуговування бібліотек екранів (традиційні операції з бібліотеками).

Одним з додаткових модулів JAM є генератор звітів. Компонування звіту здійснюється в редакторі екранів JAM. Опис роботи звіту здійснюється за допомогою спеціальної мови. Генератор звітів дозволяє визначити дані, що виводяться в звіт, угрупування виведеної інформації, форматування виведення тощо.

Програми, розроблені з використанням JAM, не вимагають виконавчих (run-time) систем і можуть бути виготовлені у вигляді модулів, що виконуються. Для цього розробник повинен мати компілятор С і редактор зв'язків. Для виготовлення промислової версії до складу JAM входить файл збірки (makefile), вихідні тексти (мовою С) ряду модулів програми та необхідні бібліотеки.

JAM містить вбудовану мову програмування JPL (JAM Procedural Language), за допомогою якої, в разі потреби, можна написати модулі, що реалізують специфічні дії. Дана мова інтерпретується, що спрощує налагодження. Існує можливість обміну інформацією між середовищем візуально побудованої програми, а не такими модулями. Крім того, в JAM реалізована можливість підключення зовнішніх модулів, написаних на будь-якій мові програмування, сумісних з проблемами функцій з мовою С.

З точки зору реалізації логіки, додатки JAM є подієво-орієнтованою системою. У JAM визначений набір подій, що включають відкриття і закриття вікон, натискання клавіші клавіатури, спрацьовування системного таймера, отримання та передача управління кожним елементом екрана. Розробник реалізує логіку програми шляхом визначення обробника кожної події. Наприклад, обробник події "натискання кнопки на екрані" (мишею або за допомогою клавіатури) може відкрити наступне екранне вікно. Обробниками подій в JAM можуть бути як вбудовані функції JAM, так і функції, написані розробником на С або JPL. Набір вбудованих функцій включає в себе більш ніж 200 функцій різного призначення. Вбудовані функції доступні для викликів з функцій, написаних як на JPL, так і на С.

Промислова версія програми, розробленої за допомогою JAM, включає в себе наступні компоненти:

- модуль інтерпретатора програми, що виконується. У цей модуль можуть бути вбудовані функції, написані розробниками на мовах 3-го покоління;

- екрани, складові самого додатку (можуть поставлятися у вигляді окремих файлів, у складі бібліотек екранів або ж бути вмонтованими в тіло інтерпретатора);

- зовнішні JPL-модулі. Можуть поставлятися у вигляді текстових файлів або в прекомпільованому вигляді, причому прекомпільовані зовнішні JPL-модулі можуть бути як у вигляді окремих файлів, так і у складі бібліотек екранів;

- файли конфігурації програми - файли конфігурації клавіатури і терміналу, файл системних повідомлень, файл загальної конфігурації.

Взаємодія з іншими засобами.

Безпосередня взаємодія з СУБД реалізується модулями JAM/DBi (Data Base interface). Способи реалізації взаємодії в JAM поділяються на два класи: ручні та автоматичні.

При ручному способі розробник програми самостійно пише запити на SQL, в яких як джерелами, так і адресатами прийому результатів виконання запиту можуть бути як інтерфейсні елементи візуально спроектованого зовнішнього рівня, так і внутрішні, невидимі для кінцевого користувача, змінні.

Автоматичний режим, реалізований менеджером транзакцій JAM, здійснюється для типових і найбільш розповсюджених видів операцій з БД, так званих QBE (Query By Example - запити за зразком), з урахуванням досить складних взаємозв’язків між

таблицями БД і автоматичним керуванням атрибутами екранних полів введення/виведення залежно від виду транзакції (читання, запис і т.д.), в якій бере участь згенерований запит.

JAM дозволяє будувати додатки для роботи більш ніж з 20 СУБД: ORACLE, Informix, Sybase, Ingres, InterBase, NetWare SQL Server, Rdb, DB2, ODBC-сумісні СУБД та ін

Відмінною рисою JAM є високий рівень перенесення додатків між різними платформами (MS DOS/MS Windows, SunOS, Solaris (i80x86, SPARC), HP-UX, AIX, VMS/Open VMS та ін.). Може знадобитися лише "перерисовка" статичних текстових полів на екранах з перекладом на українську мову при перенесенні між середовищами DOS-Windows-UNIX. Крім того, перенесення полегшується тим, що в JAM - програмах розробляються інтерфейси для віртуальних пристроїв введення/виведення, а не для фізичних.

Таким чином, при перенесенні програми з платформи на платформу, потрібно лише визначити відповідність між фізичними пристроями введення/виведення і їх логічними уявленнями для програми.

Використання SQL, як засоба взаємодії з СУБД, створює передумови для забезпечення обміну даними між СУБД. За умови переносу структури самої БД в ряді випадків додатки можуть не вимагати ніякої модифікації, за винятком ініціалізації сеансу роботи. Така ситуація може скластися в тому випадку, якщо в додатку не використовувалися специфічні для тієї чи іншої СУБД розширення SQL.

При зростанні навантаження на систему і складності завдань, що розв’язуються (розподіленість і гетерогенність ресурсів, що

використовуються, кількість одночасно підключених користувачів, складність логіки програми), застосовується триланкова модель архітектури "клієнт-сервер" з використанням менеджерів транзакцій. Компоненти JAM/TPi-Client і JAM/TPi-Server дозволяють досить просто перейти на триланкову модель. При цьому ключову роль грає модуль JAM/TPi-Server, так як основна складність впровадження триланкової моделі полягає в реалізації логіки програми в сервісах менеджерів транзакцій.

Інтерфейс JAM/CASE подібний до інтерфейсу СУБД і дозволяє здійснити обмін інформацією між репозиторієм об'єктів JAM і репозиторієм CASE-засобів аналогічно тому, як структура БД імпортується в репозиторій JAM безпосередньо з БД. Відмінність полягає в тому, що у випадку використання CASE цей обмін є двонаправленим. Крім модулів JAM/CASEi, існує також модуль JAM/CASEi Developer's Kit. За допомогою цього модуля можна самостійно розробити інтерфейс (тобто спеціалізований модуль JAM/CASEi) для конкретного CASE-засобу, якщо готового модуля JAM/CASEi для нього не існує.

Міст (інтерфейс) Silverrun-RDM <-> JAM реалізує взаємодію між CASE-засобом Silverrun і JAM (перенесення схеми бази даних та екранних форм програми між CASE-засобом Silverrun-RDM і JAM версії 7.0). Даний програмний продукт має 2 режими роботи:

- прямий режим (Silverrun-RDM-> JAM) призначений для створення об'єктів CASE-словника і елементів сховища JAM на основі подання схем у Silverrun-RDM. У цьому режимі міст дозволяє, виходячи з уявлення моделей даних інтерфейсу в Silverrun-RDM, виробляти генерацію екранів і елементів сховища JAM.

Міст перетворює таблиці та відношення реляційних схем RDM в послідовність об'єктів JAM відповідних типів. Методика побудови моделей даних інтерфейсу в Silverrun-RDM передбачає застосування механізму підсхем для прототипування екранів програми. За описом кожної з підсхем RDM міст генерує екранну форму JAM;

- зворотний режим (JAM-> Silverrun-RDM) призначений для перенесення модифікацій об'єктів CASE-словника в реляційну модель Silverrun-RDM.

Режим реінженерингу дозволяє переносити модифікації всіх властивостей екранів JAM, імпортованих раніше з RDM, в схему Silverrun. На цьому етапі для контролю цілісності бази даних не допускаються зміни схеми у вигляді додавання або видалення таблиць і полів таблиць.

Групова робота.

Ядро JAM має вбудований інтерфейс до засобів конфігураційного керування (PVCS на платформі Windows і SCCS на платформі UNIX). Під керуванням цих систем передаються бібліотеки екранів і/або репозиторії. При відсутності таких систем, JAM самостійно реалізує частину функцій підтримки групової розробки.

Використання PVCS (див. підрозділ 5.6) є кращим у порівнянні з SCCS, оскільки дозволяє організувати єдиний архів модулів проекту для всіх платформ. Так як JAM на платформі UNIX не має прямого інтерфейсу до архівів PVCS, то вибірка модулів з архіву і повернення їх в архів виробляються з використанням PVCS Version Manager. На платформі MS-Windows JAM має вбудований інтерфейс до PVCS і дії за

вибіркою/поверненням проводяться безпосередньо з середовища JAM.

Середовище функціонування.

JAM, як середовище розробки, і ПЗ, побудоване з його використанням, не є ресурсномісткими системами. Наприклад, на платформі MS-Windows достатньо мати 8MB оперативної пам'яті і 50 MB дискового простору для середовища розробки. На UNIX-платформах вимоги до апаратури визначаються самою операційною системою.

5.2. Vantage Team Builder (Westmount I-CASE) + Uniface

5.2.1. Vantage Team Builder (Westmount I-CASE)

Vantage Team Builder [4] є інтегрованим програмним продуктом, орієнтованим на реалізацію каскадної моделі ЖЦ ПЗ і підтримку повного ЖЦ ПЗ.

Структура та функції.

Vantage Team Builder забезпечує виконання таких функцій:

- проектування діаграм потоків даних, "сутність-зв'язок", структур даних, структурних схем програм і послідовностей екранних форм;

- проектування діаграм архітектури системи - SAD (проектування складу та зв'язку обчислювальних засобів, розподілу завдань системи між обчислювальними засобами, моделювання відносин типу "клієнт-сервер", аналіз використання менеджерів транзакцій і особливостей функціонування систем у реальному часі);

- генерація коду програм мовою 4GL цільової СУБД з повним забезпеченням програмного середовища та генерація SQL-коду для створення таблиць БД, індексів, обмежень цілісності і збережених процедур;

- програмування на мові С із вбудованим SQL;

- керування версіями і конфігурацією проекту;

- багатокористувацький доступ до сховища проекту;

- генерація проектної документації за стандартними та індивідуальними шаблонами;

- експорт і імпорт даних проекту у форматі CDIF (CASE Data Interchange Format).

Vantage Team Builder поставляється в різних конфігураціях залежно від СУБД, що використовуються (ORACLE, Informix, Sybase або Ingres) в проекті або засобів розробки додатків (Uniface). Конфігурація Vantage Team Builder for Uniface відрізняється від інших деякою ступінню орієнтації на спіральну модель ЖЦ ПЗ за рахунок можливостей швидкого прототипування, що надаються Uniface.

При побудові всіх типів діаграм забезпечується контроль відповідності моделей синтаксису методів, що використовуються, а також контроль відповідності однойменних елементів і їх типів для різних типів діаграм.

При побудові DFD забезпечується контроль відповідності діаграм різних рівнів декомпозиції. Контроль за правильністю верхнього рівня DFD здійснюється за допомогою матриці списків подій (ELM). Для контролю за декомпозицією складових потоків даних використовується кілька варіантів їх опису: у вигляді діаграм структур даних (DSD) або у нотації БНФ (форма Бекуса-Наура).

Для побудови SAD використовується розширена нотація DFD, що дає можливість вводити поняття процесорів, завдань і периферійних пристроїв, що забезпечують наочність проектних рішень.

При побудові моделі даних у вигляді ERD виконується її нормалізація і вводиться визначення фізичних імен елементів даних і таблиць, які будуть використовуватися в процесі генерації фізичної схеми даних конкретної СУБД. Забезпечується можливість визначення альтернативних ключів сутностей і полів, складових додаткової точки входу в таблицю (поля індексів) і потужності відносин між сутностями.

Наявність універсальної системи генерації коду, заснованої на специфікованих засобах доступу до сховища проекту, дозволяє підтримувати високий рівень виконання проектної дисципліни розробниками:

- жорсткий порядок формування моделей;
- жорстку структуру і вміст документації;
- автоматичну генерацію вихідних кодів програм тощо.

Все це забезпечує підвищення якості та надійності розроблюваних ІС.

Для підготовки проектної документації можуть використовуватися видавничі системи FrameMaker, Interleaf або Word Perfect. Структура і склад проектної документації можуть бути налаштовані відповідно до заданих стандартів. Налаштування виконується без зміни проектних рішень.

При розробці досить великої ІС вся система в цілому відповідає одному проекту, як категорії Vantage Team Builder. Проект може бути декомпозований на ряд систем, кожна з яких

відповідною щодо автономної підсистеми ІС і розробляється незалежно від інших. Надалі системи проекту можуть бути інтегровані.

Процес проектування ІС з використанням Vantage Team Builder реалізується у вигляді 4-х послідовних фаз (стадій) - аналізу, архітектури, проектування та реалізації. При цьому, закінчені результати кожної стадії повністю або частково переносяться (імпортуються) в наступну фазу.

Всі діаграми, крім ERD, перетворюються на інший тип або змінюють вигляд відповідно до особливостей поточної фази. Так, DFD перетворюються у фазі архітектури в SAD, DSD - в DTD. Після завершення імпорту логічний зв'язок з попередньою фазою розривається, тобто в діаграми можуть вноситися всі необхідні зміни.

Взаємодія з іншими засобами.

Конфігурація Vantage Team Builder for Uniface забезпечує спільне використання двох систем в рамках єдиного технологічного середовища проектування, при цьому схеми БД (SQL-моделі) переносяться в репозиторій Uniface, і навпаки, прикладні моделі, сформовані засобами Uniface, можуть бути перенесені в репозиторій Vantage Team Builder. Можливі неузгодженості між репозиторіями двох систем усуваються за допомогою спеціальної утиліти.

Розробка екранних форм в середовищі Uniface виконується на базі діаграм послідовностей форм (FSD) після імпорту SQL-моделі. Технологія розробки ІС на базі даної конфігурації показана на рисунку 5.2.1.1.

Структура сховища (що зберігається безпосередньо в цільовій СУБД) і інтерфейси Vantage Team Builder є відкритими, що дозволяє інтеграцію з будь-якими іншими засобами.

Середовище функціонування.

Vantage Team Builder функціонує на всіх основних UNIX-платформах (Solaris, SCO UNIX, AIX, HP-UX) і VMS.

Vantage Team Builder можна використовувати в конфігурації "клієнт-сервер", при цьому база проектних даних може розташовуватися на сервері, а робочі місця розробників можуть бути клієнтами.

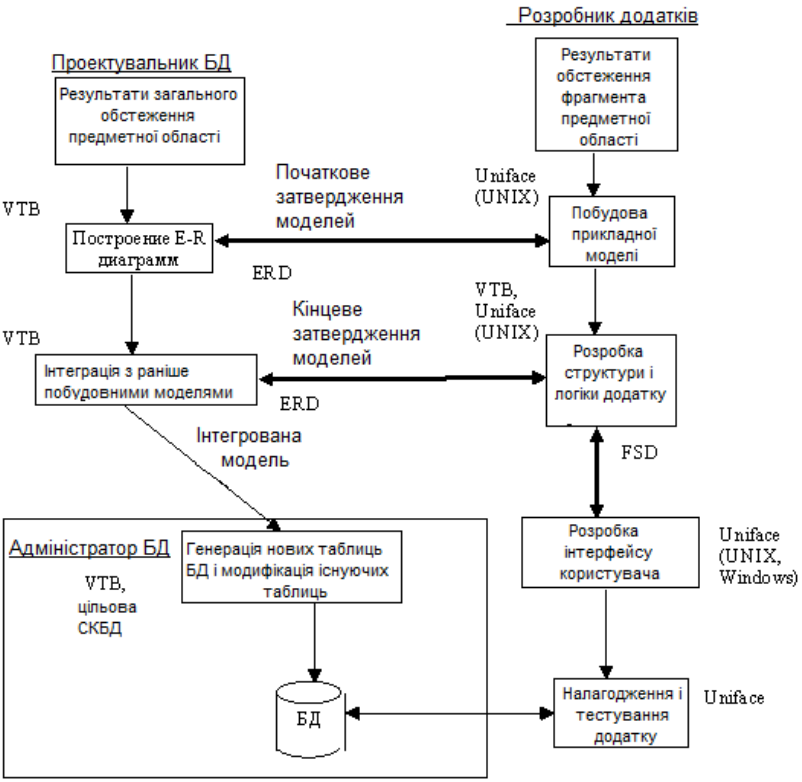

Рисунок 5.2.1.1 - Взаємодія Vantage Team Builder і Uniface

5.2.2. Uniface

Uniface 6.1 [5] - продукт фірми Compuware (США) - являє собою середовище розробки великомасштабних додатків в архітектурі "клієнт-сервер" і має наступну компонентну архітектуру:

- Application Objects Repository (репозиторій об'єктів додатків) містить метадані, які автоматично використовуються всіма іншими компонентами протягом життєвого циклу ІС (прикладні моделі опису даних, бізнес-правил, екранних форм, глобальних об'єктів і шаблонів). Репозиторій може зберігатися в будь-якій з баз даних, що підтримуються Uniface;

- Application Model Manager підтримує прикладні моделі (ER - моделі), кожна з яких представляє собою підмножину загальної схеми БД з точки зору цього додатку і включає відповідний графічний редактор;

- Rapid Application Builder - засіб швидкого створення екранних форм і звітів на базі об'єктів прикладної моделі. Він включає графічний редактор форм, засоби прототипування, налагодження, тестування і документування. Реалізовано інтерфейс з різноманітними типами віконних елементів управління (Open Widget Interface) для існуючих графічних інтерфейсів - MS Windows (включаючи VBX), Motif, OS/2. Універсальний інтерфейс подання (Universal Presentation Interface) дозволяє використовувати одну й ту ж версію програми в середовищі різних графічних інтерфейсів без зміни програмного коду;

- Developer Services (служби розробника) - використовуються для підтримки великих проектів і реалізують контроль версій (Uniface Version Control System), права доступу (розмежування повноважень), глобальні модифікації і т.д. Це забезпечує розробників засобами паралельного проектування, вхідного і вихідного контролю, пошуку, перегляду, підтримки та видачі звітів за даними системи контролю версій;

- Deployment Manager (керування поширенням додатків) - засоби, що дозволяють підготувати створений додаток для розповсюдження, встановлення і супроводження його (при цьому платформа користувача може відрізнятися від платформи розробки). До їх складу входять мережеві драйвери і драйвери СУБД, сервер додатків (полісервер), засоби поширення додатків та управління базами даних. Uniface підтримує інтерфейс практично з усіма відомими програмно-апаратними платформами, СУБД, CASE-засобами, мережевими протоколами і менеджерами транзакцій;

- Personal Series (персональні засоби) - використовуються для створення складних запитів і звітів у графічній формі (Personal Query і Personal Access - PQ/PA), а також для перенесення даних в такі системи, як WinWord і Excel;

- Distributed Computing Manager - засіб інтеграції з менеджерами транзакцій Tuxedo, Encina, CICS, OSF DCE.

Версія Uniface 7 повністю підтримує розподілену модель обчислень і триланкову архітектуру "клієнт-сервер" (з можливістю зміни схеми декомпозиції додатків на етапі виконання). Програми, що створюються за допомогою Uniface 7, можуть виконуватися в гетерогенних операційних середовищах, що використовують різні мережеві протоколи, одночасно на декількох різнорідних платформах (у тому числі і в Internet).

До складу компоненту Uniface 7 входять:

- Uniface Application Server - сервер додатків для розподілених систем;

- WebEnabler - серверне ПЗ для експлуатації додатків в Internet і Intranet;

- Name Server - серверне ПЗ, що забезпечує використання розподілених прикладних ресурсів;

- PolyServer - засіб доступу до даних і інтеграції різних систем.

У список СУБД, що підтримуються, входять DB2, VSAM і IMS. PolyServer забезпечує також взаємодію з ОС MVS.

Середовище функціонування Uniface - всі основні UNIX - платформи і MS Windows.

5.3. Designer/2000 + Developer/2000

CASE-засіб Designer/2000 2.0 фірми ORACLE [3] є інтегрованим CASE-засобом, що забезпечує у сукупності з засобами розробки додатків Developer/2000 підтримку повного ЖЦ ПЗ для систем, які використовують СУБД ORACLE.

Структура та функції.

Designer/2000 - це родина методологій і програмних продуктів, що підтримують їх.

Базова методологія Designer/2000 (CASE * Method) - структурна методологія проектування систем, що повністю охоплює всі етапи життєвого циклу ІС [8,9]. Відповідно до цієї методології на етапі планування визначаються цілі створення системи, пріоритети та обмеження, розробляється системна архітектура і план розробки ІС. У процесі аналізу будуються:

- модель інформаційних потреб (діаграма "сутність-зв’язок");

- діаграма функціональної ієрархії (на основі функціональної декомпозиції ІС);

- матриця перехресних посилань;

- діаграма потоків даних.

На етапі проектування розробляється:

- детальна архітектура ІС;

- проектується схема реляційної БД;

- програмні модулі;

- встановлюються перехресні посилання між компонентами ІС для аналізу їх взаємного впливу і контролю за змінами.

На етапі реалізації створюється:

- БД;

- будуються прикладні системи:

- проводиться їх тестування;

- перевірка якості та відповідності вимогам користувачів;

- створюється системна документація, матеріали для навчання і керівництва користувачів.

На етапах експлуатації та супроводження аналізуються продуктивність і цілісність системи, виконується підтримка і, при необхідності, модифікація ІС.

Designer/2000 забезпечує графічний інтерфейс при розробці різних моделей (діаграм) предметної області. У процесі побудови моделей, інформація про них заноситься в репозиторій. До складу Designer/2000 входять наступні компоненти:

- Repository Administrator - засоби керування репозиторієм (створення або видалення програм, керування доступом до даних з боку різних користувачів, експорт та імпорт даних);

- Repository Object Navigator - засоби доступу до сховища забезпечують багатовіконний об’єктно-орієнтований інтерфейс доступу до всіх елементів сховища;

- Process Modeller - засіб аналізу і моделювання ділової діяльності, що грунтується на концепціях реінженірингу бізнес-процесів (BPR - Business Process Reengineering) і глобальної системи керування якістю (TQM - Total Quality Management);

- Systems Modeller - набір засобів побудови функціональних та інформаційних моделей проектованої ІС, що включає CASE-засоби для побудови діаграм "сутність-зв'язок" (Entity-Relationship Diagrammer), діаграм функціональних ієрархій (Function Hierarchy Diagrammer), діаграм потоків даних (Data Flow Diagrammer) і засіб аналізу та модифікації зв'язків об'єктів репозиторію різних типів (Matrix Diagrammer);

- Systems Designer - набір засобів проектування ІС, що включає засіб побудови структури реляційної бази даних (Data Diagrammer), а також засоби побудови діаграм, що відображають взаємодію з даними; ієрархію, структуру і логіку додатків, реалізовану збереженими процедурами мовою PL/SQL (Module Data Diagrammer, Module Structure Diagrammer і Module Logic Navigator);

- Server Generator - генератор описів об'єктів БД ORACLE (таблиць, індексів, ключів, послідовностей і т.д.). Крім продуктів ORACLE, генерація і реінженеринг БД може виконуватися для СУБД Informix, DB/2, Microsoft SQL Server, Sybase, а також для стандарту ANSI SQL DDL і баз даних, доступ до яких реалізується за допомогою ODBC;

- Forms Generator (генератор програм для ORACLE Forms). Генеровані програми включають в себе різні екранні форми, засоби контролю даних, перевірки обмежень цілісності і автоматичні

підказки. Подальша робота з додатком виконується в середовищі Developer/2000;

- Repository Reports - генератор стандартних звітів, інтегрований з ORACLE Reports і дозволяє в автоматичному режимі перекласти на українську мову звіти, а також змінювати структурне подання інформації за вимогою користувачів.

Репозиторій Designer/2000 являє собою сховище всіх проектних даних і може працювати в багатокористувацькому режимі, забезпечуючи паралельне оновлення інформації кількома розробниками. У процесі проектування автоматично підтримуються перехресні посилання між об'єктами словника і можуть генеруватися більше 70 стандартних звітів про предметну область, що моделюється. Фізичне середовище зберігання репозиторію - база даних ORACLE.

Генерація додатків, крім продуктів ORACLE, виконується також для Visual Basic.

Взаємодія з іншими засобами.

Designer/2000 можна інтегрувати з іншими засобами, використовуючи відкритий інтерфейс додатків API (Application Programming Interface) ОС Windows. Крім того, можна використовувати засіб ORACLE CASE Exchange для експорту/імпорту об'єктів репозиторію з метою обміну інформацією з іншими CASE-засобами.

Developer/2000 забезпечує розробку перенесених додатків, що працюють в графічному середовищі ОС Windows, Macintosh або Motif. У середовищі ОС Windows інтеграція додатків Developer/2000 з іншими засобами реалізується через механізм OLE і керуючих елементів VBX. Взаємодія додатків з іншими СУБД

(DB/2, DB2/400, Rdb) реалізується за допомогою засобів ORACLE Client Adapter для ODBC, ORACLE Open Gateway і API.

Середовище функціонування

Середовище функціонування Designer/2000 і Developer/2000 - родина ОС Windows.

5.4. Локальні засоби (ERwin, BPwin, S-Designor, CASE-Аналітик)

ERwin - засіб концептуального моделювання БД [4], що використовує методологію IDEF1X (див. підрозділ 2.5). ERwin реалізує проектування схеми БД, генерацію її опису мовою цільової СУБД (ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server, Progress тощо) і реінжиніринг існуючої БД.

ERwin випускається в декількох різних конфігураціях, орієнтованих на найбільш поширені засоби розробки додатків 4GL. Версія ERwin/OPEN повністю сумісна із засобами розробки додатків PowerBuilder і SQLWindows і дозволяє експортувати опис спроектованої БД безпосередньо в репозиторій даних CASE-засобів.

Для ряду засобів розробки додатків (PowerBuilder, SQLWindows, Delphi, Visual Basic) виконується генерація форм і прототипу.

Мережева версія Erwin ModelMart забезпечує узгоджене проектування БД і додатків у рамках робочої групи.

BPwin - засіб функціонального моделювання, що реалізує методологію IDEF0 (див. підрозділ 2.2).

S-Designor 4.2 являє собою CASE-засіб для проектування реляційних баз даних [5]. За своїми функціональними можливостями і вартістю він близький до CASE-засобу ERwin, відрізняючись зовні нотацією, що використовується на діаграмах.

S-Designor реалізує стандартну методологію моделювання даних і генерує опис БД для таких СУБД, як ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server та ін. Для існуючих систем виконується реінжиніринг БД.

S-Designor сумісний з низкою засобів розробки додатків (PowerBuilder, Uniface, TeamWindows тощо) і дозволяє експортувати опис БД в репозиторії даних CASE-засобів. Для PowerBuilder виконується також пряма генерація шаблонів додатків.

CASE-Аналітик 1.1 [3] є практично єдиним в даний час конкурентоспроможним вітчизняним CASE-засобом функціонального моделювання та реалізує побудову діаграм потоків даних відповідно до методології, описаної в підрозділі 2.3. Його основні функції:

- побудова і редагування DFD;

- аналіз діаграм і проектних специфікацій на повноту і несуперечність;

- отримання різноманітних звітів по проекту;

- генерація макетів документів відповідно до вимог ГОСТ 19.XXX і 34.XXX.

Орієнтовна вартість:

- однокористувацька версія - 605 $;

- багатокористувацька версія (за одне робоче місце) - 535 $.

База даних проекту реалізована в форматі СУБД Paradox і є відкритою для доступу.

За допомогою окремого програмного продукту (Catherine) виконується обмін даними з CASE-засобом ERwin. При цьому з проекту, виконаного в CASE, експортується опис структур даних і накопичувачів даних, які за певними правилами формують опис сутностей та їх атрибутів.

5.5. Об'єктно-орієнтовані CASE-засоби (Rational Rose)

Rational Rose - CASE-засіб фірми Rational Software Corporation (США) - призначений для автоматизації етапів аналізу і проектування ПЗ, для генерації кодів на різних мовах і випуску проектної документації [2].

Rational Rose використовує синтез-методологію об'єктно-орієнтованого аналізу і проектування, засновану на підходах трьох провідних фахівців у цій галузі: Буча, Рамбо і Джекобсона. Розроблена ними універсальна нотація для моделювання об'єктів (UML - Unified Modeling Language) претендує на роль стандарту в області об'єктно-орієнтованого аналізу і проектування.

Конкретний варіант Rational Rose визначається мовою, на якій генеруються коди програм (C++, Smalltalk, PowerBuilder, Ada, SQLWindows і ObjectPro). Основний варіант - Rational Rose/C++ - дозволяє розробляти проектну документацію у вигляді діаграм і специфікацій, а також генерувати програмні коди на C++. Крім того, Rational Rose містить засоби реінженірингу програм, що забезпечують повторне використання програмних компонентів у нових проектах.

Структура та функції.

В основі роботи Rational Rose лежить побудова різноманітних діаграм і специфікацій, що визначають логічну й фізичну структури моделі, її статичні і динамічні аспекти. У їх число входять:

- діаграми класів;
- станів;
- сценаріїв;
- модулів;
- процесів [2].

У складі Rational Rose можна виділити 6 основних структурних компонентів:

- репозиторій;
- графічний інтерфейс користувача;
- засіб перегляду проекту (browser);
- засіб контролю проекту;
- CASE-засоби збору статистики;
- генератор документів.

До них додаються генератор кодів (індивідуальний для кожної мови) і аналізатор для C++, що забезпечує реінжиніринг - відновлення моделі проекту по вихідним текстам програм.

Репозиторій являє собою об'єктно-орієнтовану базу даних. Засоби перегляду забезпечують "навігацію" по проекту, у тому числі, переміщення по ієрархії класів і підсистем, переключення від одного типу діаграм до іншого тощо. Засоби контролю і збору статистики дають можливість знаходити і усувати помилки в міру розвитку проекту, а не після завершення його опису. Генератор

звітів формує тексти вихідних документів на основі розміщеної в репозиторії інформації.

Засоби автоматичної генерації кодів програм на мові C++, використовуючи інформацію, що міститься в логічній та фізичній моделях проекту, формують файли заголовків і файли описів класів і об'єктів. Створюваний таким чином скелет програми може бути уточнений шляхом прямого програмування мовою C++.

Аналізатор кодів C++ реалізований у вигляді окремого програмного модуля. Його призначення полягає в тому, щоб створювати модулі проектів у формі Rational Rose на основі інформації, що міститься в визначених користувачем вихідних текстах на C++. У процесі роботи аналізатор здійснює контроль правильності вихідних текстів і діагностику помилок.

Модель, отримана в результаті його роботи, може цілком або фрагментарно використовуватися в різних проектах. Аналізатор має широкі можливості налаштування по входу і виходу. Наприклад, можна визначити типи вихідних файлів, базовий компілятор, задати, яка інформація повинна бути включена в формовану модель і які елементи вихідної моделі слід виводити на екран. Таким чином, Rational Rose/C++ забезпечує можливість повторного використання програмних компонентів.

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

- діаграми класів;
- діаграми станів;
- діаграми сценаріїв;
- діаграми модулів;
- діаграми процесів;

- специфікації класів, об’єктів, атрибутів та операцій

- заготовки текстів програм;

- модель програмної системи, що розробляються.

Останній з перерахованих документів є текстовим файлом, що містить всю необхідну інформацію про проект (у тому числі необхідну для отримання всіх діаграм і специфікацій).

Тексти програм є заготовками для подальшої роботи програмістів. Вони формуються в робочому каталозі у вигляді файлів типів H (заголовки, що містять описи класів) і C_{pp} (заготовки програм для методів).

Система включає в програмні файли власні коментарі, які починаються з послідовності символів //##. Склад інформації, що включається в програмні файли, визначається або по замовчуванню, або за бажанням користувача. Надалі ці вихідні тексти розвиваються програмістами в повноцінні програми.

Взаємодія з іншими засобами і організація групової роботи.

Rational Rose інтегрується із засобом PVCS для організації групової роботи та керування проектом і з засобом SoDA - для документування проектів. Інтеграція Rational Rose і SoDA забезпечується засобами SoDA.

Для організації групової роботи в Rational Rose можлива декомпозиція моделі на керовані підмоделі. Кожна з них незалежно зберігається на диску або завантажується в модель. В якості підмоделей можуть виступати категорія класів чи підсистема.

Для керованої підмоделі передбачені операції:

- завантаження підмоделі в пам’ять;

- вивантаження підмоделі з пам’яті;

- збереження підмоделі на диску у вигляді окремого файлу;

- встановлення захисту від модифікації;

- заміна підмоделі в пам’яті на нову.

Найбільш ефективно групова робота організується при інтеграції Rational Rose зі спеціальними засобами керування конфігурацією і контролю версій (PVCS). У цьому випадку захист від модифікації встановлюється на всі керовані підмоделі, крім тих, які виділені конкретному розробникові. У цьому випадку ознака захисту від запису встановлюється для файлів, які містять підмоделі, тому при зчитуванні "чужих" підмоделей захист їх від модифікації зберігається і випадковий вплив виявляться неможливим.

Середовище функціонування.

Rational Rose функціонує на різних платформах: IBM (в середовищі родини ОС Windows), Sun SPARC stations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 (AIX).

5.6. Допоміжні засоби підтримки життєвого циклу ПЗ

5.6.1. Засоби конфігураційного управління

Мета конфігураційного управління (КУ) - забезпечити керованість і контрольованість процесів розробки та супроводження ПЗ. Для цього необхідна точна і достовірна інформація про стан ПЗ і його компонентів в кожен момент часу, а також про всіх передбачуваних і виконаних змінах.

Для вирішення завдань КУ застосовуються методи і засоби, які забезпечують:

- ідентифікацію стану компонентів;
- облік номенклатури всіх компонентів і модифікацій системи в цілому;
- контроль за змінами, що вносяться в компоненти;
- структуру системи та її функції;
- координоване управління розвитком функцій;
- поліпшенням характеристик системи.

Найбільш поширеним засобом КУ є PVCS фірми Intersolv (США), що включає ряд самостійних програмних продуктів: PVCS Version Manager, PVCS Tracker, PVCS Configuration Builder і PVCS Notify.

Програмний продукт PVCS Version Manager [8] призначений для управління всіма компонентами проекту і ведення планомірної

багатоверсійності і багатоплатформенності розробки силами команди розробників в умовах однієї або кількох локальних мереж.

Поняття "проект" трактується як сукупність файлів. У процесі роботи над проектом проміжний стан файлів періодично зберігається в архіві проекту, ведуться записи про час збереження, відповідно один одному декількох варіантів різних файлів проекту. Крім цього, фіксуються імена розробників, відповідальних за той чи інший файл, склад файлів проміжних версій проекту та ін. Це дозволяє повернутися за необхідністю до будь-якого з попередніх станів файлу (наприклад, при виявленні помилки, яку в даний момент важко виправити).

PVCS Version Manager призначений для використання в робочих групах. Система блокувань, реалізована в PVCS Version Manager, дозволяє запобігти одночасному внесенню змін в один і той же файл. У той же час, PVCS Version Manager дозволяє розробникам працювати з власними версіями загального файлу з напівавтоматичним врегулюванням конфліктів між ними.

Доступ до архівів PVCS Version Manager можливий не тільки через сам Version Manager, а й з більш ніж 50 інструментальних засобів, у тому числі: MS Visual C і MS Visual Basic, Uniface, PowerBuilder, SQL Windows, JAM, Delphi, Paradox та ін.

Результатом роботи програмного продукту PVCS Version Manager є створений засобами файлової системи репозиторій, який зберігає в компактній формі всі робочі версії програмного продукту разом з необхідними коментарями та мітками.

PVCS Version Manager функціонує в середовищі родини ОС Windows, OS/2, SunOS, Solaris, HP-UX, AIX і SCO UNIX і може виконуватися на будь-якому персональному комп’ютері з

процесором 80386 або вище, робочих станціях Sun, HP і IBM (RS-6000).

Іншим засобом конфігураційного управління є PVCS Tracker [8] - спеціалізована надбудова над офісною електронною поштою, призначена для обробки повідомлень про помилки в роботі програмного продукту, доведення інформації про виявлені помилки та їх характер до відома користувача/розробника пакету і контролю за виконанням.

Інтеграція з PVCS Version Manager дає можливість пов'язувати з повідомленнями ті чи інші компоненти проекту. Звітні можливості PVCS Tracker включають безліч різновидів графіків і діаграм, що відображають стан проекту і процесу його налагодження, зрізи по різним компонентам проекту розробникам і тестувальникам. З їх допомогою можна наочно показати поточний стан роботи над проектом та його тимчасові тенденції.

Персонал, який працює з PVCS Tracker ділиться на п'ять груп залежно від їх обов'язків:

- користувачі;

- розробники;

- група тестування і контролю якості;

-- група технічної підтримки і супроводу;

- керуючий персонал.

Цим п'яти групам персоналу відповідають п'ять зумовлених груп PVCS Tracker:

- користувачі (Submitters) - мають обмежені права на внесення зауважень та повідомлень про помилки в базу даних PVCS Tracker;

- розробники (Development Engineers) - мають право розробляти основні операції з вимогами та зауваженнями в базі даних PVCS Tracker. Якщо розробники діляться на підгрупи, то для кожної підгрупи можуть бути задані окремі списки прав доступу;

- тестувальники (Quality Engineers) - мають право розробляти основні операції з вимогами та зауваженнями;

- супровід (Support Engineers) - мають право вносити будь-які зауваження, вимоги та рекомендації в базу даних, але не мають прав з розподілу роботи і зміни її пріоритетності та термінів виконання;

- керівники (Managers) - мають право розподіляти роботу між виконавцями і приймати рішення про їх належне виконання. Керівникам різних груп можуть надавати різні права доступу до бази даних PVCS Tracker безпосередньо виконавцям проектів в своїх групах/підрозділах.

На додаток до цих п’яти, наперед визначених груп, існує група адміністратора бази даних і 11 додаткових груп, які можуть бути налаштовані у відповідності зі специфічними посадовими обов’язками співробітників, що використовують PVCS Tracker.

Вимога або зауваження, що надходять в PVCS Tracker, проходить чотири етапи обробки:

- реєстрація - внесення зауваження до бази даних;

- розподіл - призначення відповідального виконавця і термінів виконання;

- виконання - усунення зауваження, яке в свою чергу, може викликати додаткові зауваження чи вимоги на додаткові роботи;

- приймання - приймання робіт та зняття їх з контролю або направлення на доопрацювання.

Вимоги та зауваження, що надходять до бази даних, PVCS Tracker оформляються у вигляді спеціальної форми, яка може містити до 18 полів вибору стандартних значень і до 12 довільних текстових рядків. При розробці форми слід визначити оптимальний набір інформації, характерний для всіх записів в базі даних.

Для отримання змістовної інформації про хід розробки PVCS Tracker дозволяє отримувати три типи статистичних звітів:

- частотні;
- тренди;
- діаграми розподілу.

Частотні звіти містять інформацію про частоту зауважень за одну годину тестування програмного продукту. Однак універсального частотного звіту не існує, тому що на оцінку якості впливають тип методів тестування, серйозність виявлених помилок і значення дефектних модулів для функціонування всієї системи. Мале число фатальних помилок, що призводять до повної зупинки розробки, гірше великого числа зауважень до зовнішнього вигляду інтерфейсу користувача. Отже, частотні звіти повинні бути налаштовані на виявлення якого-небудь конкретного аспекту якості для того, щоб їх можна було використовувати для прогнозування закінчення робіт над проектом.

Тренди містять інформацію про зміни того чи іншого показника в часі і характеризують стабільність і безперервність процесу розробки. Вони дозволяють відповісти на питання:

- чи встигає група розробників впоратися зі вступними зауваженнями;

- чи поліпшується якість програмного продукту і яка динаміка цього процесу;

- як вплинуло те чи інше рішення (збільшення числа розробників, введення змінного графіка, впровадження нового методу тестування) на роботу групи тощо.

Діаграми розподілу - найбільш різноманітні і корисні для здійснення оперативного керівництва форми звітів. Вони дозволяють відповісти на питання: який метод тестування більш ефективний, які модулі викликають найбільше число нарікань, хто з розробників краще справляється з конкретним типом завдань, чи немає перекосу у розподілі робіт між виконавцями, чи немає модулів, тестуванням яких було приділено недостатньо уваги і т . д.

PVCS Tracker призначений для використання в робочих групах, об'єднаних в загальну мережу. У цьому випадку центральна база або проект PVCS Tracker знаходиться на загальнодоступному сервері мережі, доступ до якого реалізується за допомогою ODBC-драйверів, що входять до складу PVCS Tracker.

Головною особливістю PVCS Tracker, порівняно зі звичайним додатком СУБД, є його здатність автоматично повідомляти користувача про надходження цікавої або належної до його компетенції інформації та гнучка система розподілу повноважень всередині робочої групи. При необхідності, PVCS Tracker може використовувати для повідомлення віддалених у просторі членів групи через електронну пошту:

- PVCS Tracker підтримує групову роботу в локальних мережах і взаємодіє з СУБД dBase, ORACLE, SQL Server і SYBASE допомогою ODBC.

- PVCS Tracker може бути інтегрований з будь-якою системою електронної пошти, що підтримує стандарти VIM, MAPI або SMTP.

- PVCS Version Manager і PVCS Tracker оточені допоміжними компонентами: PVCS Configuration Builder і PVCS Notify.

- PVCS Configuration Builder призначений для складання остаточного продукту з компонентів проекту. PVCS Configuration Builder дозволяє описувати процес складання як на стандартній мові MAKE, так і на власній внутрішній мові, що має істотно більші можливості. PVCS Configuration Builder дозволяє здійснювати збірку програмного продукту на підставі файлів, що зберігаються в репозиторії PVCS Version Manager.

Звичайна процедура збирання програмного продукту за допомогою PVCS Configuration Builder складається з трьох кроків:

- будується файл залежностей між вихідними модулями;

- в отриманий файл вносяться зміни з метою його налаштування і оптимізації;

- здійснюється складання програмного продукту з вихідних модулів.

Результатом роботи PVCS Configuration Builder є спеціальний файл, що описує оптимальний алгоритм збирання програмного продукту, побудований на основі аналізу дерева залежностей між вихідними модулями.

PVCS Notify забезпечує автоматичну розсилку повідомлень про помилки з бази даних пакету PVCS Tracker по робочим станціям призначення. При цьому використовується офісна система

електронної пошти: Mail або Microsoft Mail. PVCS Notify розширює можливості PVCS Tracker і використовується тільки разом з ним.

PVCS Notify налаштовується з середовища PVCS Tracker. Налаштування включає в себе визначення інтервалу часу, через який PVCS Notify перевіряє вміст бази даних, визначення критеріїв відбору записів для розсилки повідомлень, визначення списків адрес для розсилки. Після налаштування PVCS Notify починає роботу в автономному режимі, автоматично розсилаючи повідомлення про зміни в базі даних PVCS Tracker.

PVCS Notify призначений для використання у великих робочих групах, частина членів яких хоча і доступна тільки через засоби електронної пошти, однак повинна мати оперативну інформацію про вимоги на зміну програмного продукту, зауваження, помилки, хід і результати його тестування.

Результатом роботи PVCS Notify є оформлені відповідно до одного із стандартів поштові повідомлення, готові для розсилки за допомогою системи електронної пошти.

5.6.2. Засоби документування

Для створення документації в процесі розробки ІС використовуються різноманітні засоби формування звітів, а також компоненти видавничих систем. Зазвичай засоби та обладнання для документування вбудовані в конкретні CASE-засоби. Винятком є деякі пакети прикладних програм, що мають додатковий сервіс при документуванні. З них найбільш активно використовується SoDA (Software Document Automation).

Програмний продукт SoDA призначений для автоматизації розробки проектної документації на всіх фазах ЖЦ ПЗ. Він дозволяє автоматично отримувати різноманітну інформацію, що формується на різних стадіях розробки проекту, і включати її у перелік вихідних документів. При цьому:

- контролюється відповідність документації проекту;
- контролюється взаємозв'язок документів;
- забезпечується їх своєчасне оновлення;
- результуюча документація автоматично формується з багатьох джерел, число яких не обмежене.

SoDA не залежить від інструментів, що застосовуються. Зв'язок з додатками здійснюється через стандартний програмний інтерфейс API. Перехід на нові інструментальні засоби не тягне за собою додаткових витрат з документування проекту.

SoDA містить набір шаблонів документів, визначених стандартом на програмне забезпечення DOD 2167A. На їх основі можна без спеціального програмування створювати нові форми документів, що визначаються користувачами.

Пакет включає в себе графічний редактор для підготовки шаблонів документів. Він дозволяє задавати необхідний стиль, фон, шрифт, визначати розташування заголовків, резервувати місця, де розміщуватиметься здобута із різноманітних джерел інформація. Зміни автоматично вносяться тільки в ті частини документації, на які вони вплинули в програмі. Це скорочує час підготовки документації за рахунок відмови від перегенерації всієї документації.

SoDA реалізована на базі видавничої системи FrameBuilder і надає повний набір засобів для редагування і верстки документації, що випускається. Різні версії документації можуть бути для наочності відзначені своїми відмінними ознаками.

В системі створюються таблиці вимог до проекту, за якими можна простежити, як реалізуються ці вимоги. Різні види документації, які супроводжують різні етапи ЖЦ, пов'язані між собою, і можна простежити стан:

- проекту від первинних вимог до аналізу;
- проектування;
- кодування;
- тестування програмного продукту.

Підсумковим результатом роботи системи SoDA є готовий документ (або книга). Документ може зберігатися у файлі формату SoDA (Frame Builder), який виходить в результаті генерації документа. Виведення на друк цього документа (або його частини) можливий із системи SoDA.

Середовище функціонування SoDA - ОС типу UNIX на робочих станціях Sun SPARCstation, IBM RISC System/6000 або Hewlett Packard HP 9000 700/800.

5.6.3. Засоби тестування

Під тестуванням розуміється процес пробного виконання програми з метою виявлення помилок. Регресивне тестування - це тестування, що проводиться після вдосконалення функцій програми або внесення до неї змін.

Один з найбільш розвинених засобів тестування QA (нова назва - Quality Works) [2] є інтегрованим, багатоплатформеним середовищем для розробки автоматизованих тестів будь-якого рівня, включаючи тести регресії для додатків із графічним інтерфейсом користувача.

QA дозволяє:

- починати тестування на будь-якій фазі ЖЦ;
- планувати процес тестування;
- керувати процесом тестування;
- відображати зміни в додатку;
- повторно використовувати тести для більш, ніж 25 різних платформ.

Основними компонентами QA є:

- QA Partner - середовище для розробки, компіляції та виконання тестів;
- QA Planner - модуль для розробки планів тестування та обробки результатів. Для створення і виконання тестів в процесі роботи, QA Planner викликається QA Partner;
- Agent - модуль, що підтримує роботу пакету в мережі.

Процес тестування складається з наступних етапів:

- створення плану тестування;

- зв’язування плану з тестами;

- позначка і виконання тестів;

- отримання звітів про тестування і керування результатами.

Створення тестового плану в QA Planner включає в себе складання схеми тестових вимог і виділення рівнів деталізації. Для цього необхідно визначити весь обсяг тестування програмних продуктів, підготувати функціональну декомпозицію програми, оцінити, скільки тестів необхідно для кожної функції і характеристики, визначити, скільки з них буде реалізовано в залежності від доступних ресурсів і часу. Ця інформація використовується для створення схеми тестових вимог.

Для зв’язування плану з тестами необхідно розробити керуючі пропозиції (скрипти) на спеціальній мові 4Test і тести, які виконають вимоги плану і зв’язати компоненти будь-яким способом. Для уникнення перевантаження тестів використовують керування тестовими даними.

При виконанні плану результати записуються у форматі, близькому/схожому на формат плану. Всі результати пов’язані з планом. Є можливість переглянути або приховати загальну інформацію про виконання, злити файли результатів, розмітити невдалі тести, порівняти результати попереднього виконання тестів, виконати або відмінити звіт.

Одним з атрибутів тесту є ім’я його розробника, що дозволяє при необхідності виконувати тести, створені конкретним розробником.

Комплекс QA займає на жорсткому диску не більше 21Мб. Підтримувані платформи: ОС родини Windows, OS / 2, Macintosh, VMS, HP-UX, AIX, Solaris.

5.7. Приклади комплексів CASE-засобів

На закінчення приведемо приклади комплексів CASE-засобів, які забезпечують підтримку повного ЖЦ ПЗ. Тут хотілося б ще раз відзначити недоцільність порівняння окремо взятих CASE-засобів, оскільки жоден з них не вирішує в цілому всі проблеми створення та супроводження ПЗ. Це підтверджується повним набором критеріїв оцінки і вибору, які зачіпають всі етапи ЖЦ ПЗ.

Порівнюватися можуть комплекси методологічно і технологічно узгоджених інструментальних засобів, що підтримують повний ЖЦ ПЗ і забезпечені необхідною технічною та методичною підтримкою з боку фірм-постачальників. На думку авторів, на сьогоднішній день найбільш розвиненим зі всіх, що поставляються в Україні комплексів такого роду, є комплекс технологій та інструментальних засобів створення ІС, заснований на методології та технології DATARUN. До складу комплексу входять наступні інструментальні засоби:

- CASE-засіб Silverrun;
- засіб розробки додатків JAM;
- міст Silverrun-RDM <-> JAM;
- комплекс засобів тестування QA;
- менеджер транзакцій Tuxedo;
- комплекс засобів планування та керування проектом SE Companion;
- комплекс засобів конфігураційного керування PVCS;
- об’єктно-орієнтований CASE-засіб Rational Rose;
- засіб документування SoDA.

Прикладами інших подібних комплексів є:

- Vantage Team Builder for Uniface + Uniface (фірми "DataX/Florin" і "ЛАНІТ");

- CASE-засоби Designer/2000 (основний), ERwin, Bpwin і Oowin (альтернативні);

- засоби розробки додатків Developer/2000, ORACLE Power Objects (основні) і Usoft Developer (альтернативний);

- засіб налаштування та оптимізації ExplainSQL (Platinum);

- засоби адміністрування та супроводження SQLWatch, DBVision, SQL Spy, TSReorg та ін (Platinum);

- засіб документування ORACLE Book.

- комплекс засобів на основі продуктів фірми CENTURA:

- CASE-засоби ERwin, Bpwin і Oowin (об'єктно-орієнтований аналіз);

- засоби розробки додатків SQLWindows і TeamWindows;

- засіб тестування та оптимізації програм "клієнт-сервер" SQLBench (ARC);

- засоби експлуатації та супроводження Quest і Crystal Reports.

Перелік скорочень, символів, спеціальних термінів

БД - база даних

ДПД - діаграма потоків даних

ДСД - діаграма структур даних

ЖЦ - життєвий цикл

ЖЦ ПЗ - життєвий цикл програмного забезпечення

ІС - інформаційна система

ІСАМ - інтеграція комп'ютерних та промислових технологій

КІ - комп'ютерна інженерія

КС - комп'ютерна система

КУ - конфігураційне управління

НСІ - носії інформації

ОП - об'єкт проектування

ПЗ - програмне забезпечення

ПК - персональний комп'ютер

САПР - система автоматизованого проектування

СУБД - система управління базами даних

ТС - тимчасове керування;

TD - тимчасові дані;

TCD - тимчасове керування/дані

ЦНТУ - Центральноукраїнський національний технічний університет

Термінологічний словник

Виміри (Dimensions) — осі багатовимірного гіперкуба, у ролі яких виступають основні атрибути аналізованого бізнес-процесу.

Гіперкуб (куб, Cube) — багатовимірна модель даних, що є інтуїтивно зрозумілою і використовується для аналізу ділової інформації.

Життєвий цикл ПЗ — модель створення і супроводження ПЗ, що відображає його різні стани, починаючи з моменту виникнення ідеї створення ПЗ і закінчуючи моментом виходу його з експлуатації.

Міри (Measures) — дані, що кількісно характеризують процес і знаходяться всередині гіперкуба на перетинаннях гіперплощин, що відповідають міткам.

Мітки (члени виміру, Members) — значення, що відкладаються вздовж виміру. Мітки використовуються як для «розрізування» куба, так і для обмеження (фільтрації) даних. Мітки можуть утворювати ієрархії.

Репозиторій (repository) — база даних CASE, в якій зберігається вся проектна інформація

Сховище даних (Data Warehouse) — предметно-орієнтоване, прив'язане до часу і незмінне зібрання даних для підтримки процесу управлінських рішень. Дані у сховище потрапляють з оперативних систем (OLTP-систем), призначених для автоматизації бізнес-процесів. Крім того, сховище даних може поповнюватись із зовнішніх джерел, наприклад, статистичних звітів

CASE (Computer-Aided Software/System Engineering) — сукупність методологій аналізу, проектування, розроблення і супроводження складних систем програмного забезпечення, підтримана комплексом взаємозв'язаних засобів автоматизації.

DFD (Data Flow Diagram) — діаграма потоків даних. На DFD відображаються потоки даних, процеси перетворення вхідних потоків на вихідні, сховища інформації, джерела і споживачі інформації, зовнішні щодо системи. Надалі ці діаграми є підґрунтям для формування структури розроблюваного ПЗ.

ERD (Entity-Relationship Diagram) — діаграма «сутність — зв'язок». На ERD показують так звані сутності (інформаційні об'єкти, що становлять інтерес з погляду наступного зберігання) і зв'язки між сутностями з відображенням характеру зв'язку. Ці діаграми є основою для проектування баз даних.

SC (Structure Chart) — структурна карта. SC слугують для відображення архітектури системи у вигляді сукупності програмних модулів і зв'язків між ними, а також даних, що передаються від одного модуля до іншого.

STD (State Transition Diagram) — діаграма переходів станів. На STD відображуються стани, в яких може перебувати система, і можливі переходи з одного стану до іншого. Згідно з діаграмою проектується інтерфейс користувача.

FAST (швидкий) — означає, що система повинна забезпечувати видачу більшості відповідей користувачам у межах приблизно п'ятьох секунд. При цьому найпростіші запити опрацьовуються протягом однієї секунди і дуже небагато — понад 20 секунд. Постачальники вдаються до широкого розмаїття методів, щоб досягти цієї мети, включаючи спеціалізовані форми

збереження даних, великі попередні обчислення або ж підсилюючи апаратні вимоги.

ANALYSIS (аналіз) — означає, що система може справлятися з будь-яким логічним і статистичним аналізом, характерним для даної програми, і надає його можливості у вигляді, доступному для кінцевого користувача. Необхідно дозволити користувачу визначати нові спеціальні обчислення як частину аналізу і формувати звіти будь-яким бажаним засобом, без необхідності програмування. Засоби аналізу могли б включати певні процедури типу аналізу часових рядів, розподілу витрат, валютних переведень, пошуку цілей, зміни багатовимірних структур, непроцедурного моделювання, виявлення виняткових ситуацій, витягів даних та інші операції, залежні від програми.

SHARED (розділювача) — означає, що система здійснює усі вимоги захисту конфіденційності (можливо до рівня комірки) і, якщо множинний доступ для запису необхідний, забезпечує блокування модифікацій на відповідному рівні. Не в усіх програмах існує необхідність зворотного запису даних. Проте кількість таких програм збільшується, і система повинна бути спроможна опрацювати множинні модифікації безпечним засобом. Це — головне дошкульне місце багатьох OLAP-продуктів, які мають тенденцію припускати, що в усіх програмах OLAP потрібне тільки читання, і надають спрощені засоби захисту.

MULTIDIMENSIONAL (багатовимірна) — система повинна забезпечити багатовимірне концептуальне представлення даних, включаючи повну підтримку для ієрархій і множинних ієрархій, оскільки це найбільш логічний спосіб аналізувати бізнес організації.

INFORMATION (інформація) — необхідна інформація має бути отримана там, де в ній є потреба. Потужність різноманітних продуктів може бути вимірювана тим, скільки вхідних даних вони можуть опрацьовувати, а не скільки гігабайт вони можуть зберігати. Потужність продуктів дуже різноманітна — найбільші OLAP-продукти можуть оперувати, принаймні, в тисячу разів більшою кількістю даних у порівнянні із найменшими. При цьому варто враховувати багато факторів, включаючи дублювання даних, необхідну оперативну пам'ять, використання дискового простору, експлуатаційні показники, інтеграцію з інформаційними сховищами і т. ін

OLAP (OnLine Analytical Processing) — оперативний аналіз даних. Сукупність засобів багатовимірного аналізу даних, накопичених у сховищі.

Computer Aided of Logistics Support — автоматизовані логістичні системи.

Computer Aided Acquisition and Support — автоматизовані постачання і підтримка.

Computer Aided Acquisition and Life-Cycle Support — автоматизація безперервних постачань і життєвого циклу.

Commerce At Life Speed — бізнес у високому темпі.

Information Engineering — інформаційна інженерія.

Artificial Intelligence — штучний інтелект.

Re - engineering — зворотне проектування.

Business Process Reengineering — реінжиніринг бізнес-процесів.

Multi - Agent Systems — багатоагентні системи.

Knowledge Management — управління знаннями.

Industrial Engineering — промислова інженерія.

Total Quality Management — управління якістю.

Computer Aided Software Engineering — CASE –технології.

Just - in - time — точно вчасно.

OPT (Optimised Production Technology — оптимізаційна технологія виробництва.

CIM (Computer Integrated Manufacturing — інтегровані виробництва на основі обчислювальної техніки.

CALS (Continuous Acquisition and Life Circle Support — підтримка безперервного життєвого циклу продукції).

ERP, MRP (Material Requirements Planning) — планування потреб в матеріалах.

MRP II (Manufacturing Recourse Planning) — планування ресурсів виробництва.

Total Quality Management — технологія розробки систем якості.

Information and Communication Technologies — інформаційні технології.

Список використаної літератури

1.Вендров А.М. Один из подходов к выбору средств проектирования баз данных и приложений. //СУБД, 1995, №3.

2.Зиндер Е. 3. Бизнес-реинжиниринг и технологии системного проектирования. Учебное пособие: М.: Центр Информационных Технологий, 1996.-236 с.

3.Калянов Г. Н. CASE. Структурный системный анализ (автоматизация и применение). М.: Лори, 1996. - 318 с.

4.Марка Д., Макгоуэн К. Методология структурного анали за и проектирования (SADT) / Пер. с англ. – М: МетаТехнология, 1993. – 240 с.

5.Липаев В.В. Международные стандарты, поддерживающие жизненный цикл программных средств. – М.: МП Экономика. 1996.

6.Создание информационной системы предприятия. "Computer Direct", 1996, №2.

7.Шлеер С., Меллор С. Объектно-ориентированный анализ: моделирование мира в состояниях. Пер. с англ. — Киев : Диалектика, 1993. — 240 с: ил. ISBN 0-13-629940-7 (англ.).

8.Barker R. CASE*Method. Entity-Relationship Modelling. Copyright Oracle Corporation UK Limited, Addison-Wesley Publishing Co., 1990.

Навчальний посібник

Савеленко Олена Костянтинівна
Лисенко Ірина Анатоліївна
Іванченко Олександр Олександрович

CASE-ТЕХНОЛОГІЇ У ПРОЕКТУВАННІ ІНФОРМАЦІЙНИХ СИСТЕМ

Українською мовою

Редактор: Савеленко О.К.
Технічний редактор Лисенко В.Ф.

Формат 60х84/16. Ум. друк. арк. 13,95.
Обл. вид. арк. 7,02. Тираж 300 пр. Зам. № 346.

Видавець і виготовлювач СПД ФО Лисенко В.Ф.
вул. Пацаєва, 14, корп. 1, кв. 101. м. Кропивницький,
Україна, 25030

Свідоцтво суб'єкта видавничої справи ДК №3904
від 22.10.2010