Тема 10 CASE-ЗАСОБИ ТА ВІЗУАЛЬНЕ МОДЕЛЮВАННЯ

10.1. ПЕРЕДУМОВИ ПОЯВИ CASE ЗАСОБІВ

Тенденції розвитку сучасних інформаційних технологій спричиняють постійне ускладнення автоматизованих систем (АС). Для боротьби зі складністю проектів нині створено системи автоматизованого проектування (САПР) самих програмних проектів.
Для успішної реалізації проекту об'єкт проектування (АС) має бути насамперед адекватно описаний, мають бути побудовані повні, а також несуперечливі функціональні та інформаційні моделі АС. Накопичений на сьогодні досвід проектування АС показує, що це трудомістка і тривала робота.
Все це сприяло появі програмно-технологічних засобів особливого призначення - CASE-коштів, що реалізують CASE-інженерію створення та супроводу АС. Термін "CASE" (Computer Aided Software Engineering) використовується в даний час у широкому сенсі. Початкове значення терміна CASE, обмежене питаннями автоматизації розробки тільки програмного забезпечення (ПЗ), в даний час набуло нового змісту, що охоплює процес розробки складних АС в цілому.
Тепер під терміном «CASE-засоби» розуміються програмні засоби, що підтримують процеси створення та супроводу АС, включаючи аналіз та формулювання вимог, проектування прикладного ПЗ (додатків) та баз даних, генерацію коду, тестування, документування, забезпечення якості, конфігураційне управління та управління проектом , і навіть інші процеси. CASE-засоби разом із системним ПЗ та технічними засобами утворюють середовище розробки АС.
CASE-технологіяє методологією проектування АС, а також набір інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки та супроводу АС та розробляти додатки відповідно до інформаційних потреб користувачів. Більшість існуючих CASE-засобів засновані на методологіях структурного (в основному) або об'єктно-орієнтованого аналізу та проектування, що використовують специфікації у вигляді діаграм або текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поведінки системи та архітектури програмних засобів.
На підставі анкетування понад тисячу американських фірм фірмою Systems Development Inc. у 1996 р. було складено огляд передових технологій (Survey of Advanced Technology). Згідно з цим оглядом CASE-технологія наразі потрапила до розряду найбільш стабільних інформаційних технологій (її використала половина всіх опитаних користувачів більш ніж у третині своїх проектів, з них 85% завершилися успішно). Однак, незважаючи на всі потенційні можливості CASE-засобів, існує безліч прикладів їх невдалого впровадження. У зв'язку з цим слід зазначити таке:
- CASE-засоби не обов'язково дають негайний ефект, він може бути отриманий тільки через якийсь час;
— реальні витрати на використання CASE-коштів зазвичай набагато перевищують витрати на їх придбання;
— CASE-кошти забезпечують можливості для отримання суттєвої вигоди лише після успішного завершення процесу їх впровадження.
Для успішного впровадження CASE-засобів організація повинна мати такі якості, як:
• технологія – розуміння обмеженості існуючих можливостей та здатність прийняти нову технологію;
• культура – готовність до впровадження нових процесів та взаємовідносин між розробниками та користувачами;

• управління — чітке керівництво та організованість по відношенню до найважливіших етапів та процесів впровадження.

10.2. ОГЛЯД CASE-СИСТЕМ

На сьогоднішній день ринок програмного забезпечення має у своєму розпорядженні наступні найбільш розвинені CASE-засоби:
• Vantage Team Builder (Westmount I-CASE);
• Designer/2000;
• Silverrun;
• ERwin+BPwin;
• S-Designor;
• CASE.Аналітик;
• Rational Rose.
Крім того, на ринку постійно з'являються як нові для вітчизняних користувачів системи, так і нові версії та модифікації цих систем.
CASE-засіб Silverrun американської фірми "Computer Systems Advisers, Inc." (CSA) використовується для аналізу та проектування АС бізнес-класу та орієнтовано переважно на спіральну модель життєвого циклу. Воно застосовується для підтримки будь-якої методології, заснованої на окремій побудові функціональної та інформаційної моделей (діаграм потоків даних та діаграм «сутність-зв'язок»).
Платою за високу гнучкість та різноманітність образотворчих засобів побудови моделей є така вада Silverrun, як відсутність жорсткого взаємного контролю між компонентами різних моделей (наприклад, можливості автоматичного поширення змін між ДПД різних рівнів декомпозиції).
Для автоматичної генерації схем баз даних у Silverrun існують мости до найпоширеніших СУБД: Oracle, Informix, DB2, SQL Server та ін. Для передачі даних у засоби розробки додатків є мости до мов: , Delphi.
Всі мости дозволяють завантажити в Silverrun RDM інформацію з каталогів відповідних СУБД або 4GL.
Vantage Team Builderє інтегрованим програмним продуктом, орієнтованим на реалізацію каскадної моделі життєвого циклу (ЖЦ) ПЗ та підтримку повного життєвого циклу ПЗ.
Система забезпечує виконання таких функцій:
- Проектування діаграм потоків даних, «сутність-зв'язок», структур даних, структурних схем програм та послідовностей екранних форм;
— проектування діаграм архітектури системи SAD (проектування складу та зв'язку обчислювальних засобів, розподіл завдань системи між обчислювальними засобами, моделювання відносин типу «клієнт-сервер», аналіз використання менеджерів транзакцій та особливостей функціонування систем в реальному часі);
- генерацію коду програм мовою 4GL цільової СУБД з повним забезпеченням програмного середовища та генерація SQL-коду для створення таблиць БД, індексів, обмежень цілісності та збережених процедур;
- програмування мовою С із вбудованим сервером SQL;
- Управління версіями та конфігурацією проекту;
- генерацію проектної документації за стандартними та індивідуальними шаблонами.
Vantage Team Builder функціонує на всіх основних UNIX-платформах (Solaris, SCO UNIX, AIX, HP-UX) та VMS.
CASE-засіб Designer/2000 2.0 фірми «ORACLE» є інтегрованим CASE-засобом, що забезпечує разом із засобами розробки додатків Developer/2000 підтримку повного життєвого циклу ПЗ для систем, що використовують СУБД ORACLE.
Базова методологія Designer/2000 (CASE Method) - структурна методологія проектування систем, що повністю охоплює всі етапи життєвого циклу АС. Designer/2000 забезпечує графічний інтерфейс розробки різних моделей (діаграм) предметної області. У процесі побудови моделей інформація про них заноситься до репозитарію.
Середовище функціонування Designer/2000 – Windows xt та ін.
ERwin - Засіб концептуального моделювання БД, що використовує методологію IDEF1X. ERwin реалізує проектування схеми БД, генерацію її опису мовою цільової СУБД (ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server, Progress та ін.) та реінженіринг існуючої БД. ERwin випускається в

різних конфігураціях, орієнтованих на найбільш поширені засоби розробки додатків 4GL. Версія ERwin/OPEN повністю сумісна з засобами розробки програм PowerBuilder і SQLWindows і дозволяє експортувати опис спроектованої БД безпосередньо до репозитарію даних коштів.
Для ряду засобів розробки додатків (PowerBuilder, SQLWindows, Delphi, Visual Basic) виконується генерація форм та прототипів додатків.
Мережа версія Erwin ModelMart забезпечує узгоджене проектування БД та додатків у робочій групі.
BPwin - Засіб функціонального моделювання, що реалізує методологію IDEF0.
S-Designor 4.2 є CASE-засіб для проектування реляційних баз даних. За своїми функціональними можливостями і вартістю він близький до CASE-засобу ERwin, відрізняючись нотацією, що зовні використовується на діаграмах. S-Designor реалізує стандартну методологію моделювання даних та генерує опис БД для таких СУБД, як ORACLE, Informix, Ingres, Sybase, DB/2, Microsoft SQL Server та ін. Для існуючих систем виконується реінженіринг БД.
S-Designor сумісний із низкою засобів розробки додатків (PowerBuilder, Uniface, TeamWindows та інших.) і дозволяє експортувати опис БД у репозитарії даних. Для PowerBuilder виконується пряма генерація шаблонів програм.
CASE.Аналітик 1.1є практично єдиним у цей час конкурентоспроможним вітчизняним CASE-засобом функціонального моделювання. Його основні функції:
• аналіз діаграм та проектних специфікацій на повноту та несуперечність;
• отримання різноманітних звітів щодо проекту;
• генерація макетів документів відповідно до вимог ГОСТ 19.XXX та 34.XXX.
Середовище функціонування: процесор 386 і вище, основна пам'ять 4 Мб, дискова пам'ять 5 Мб, MS Windows хт та ін.
За допомогою окремого програмного продукту (Catherine) виконується обмін даними із CASE-засобом ERwin. При цьому з проекту, виконаного в CASE. Аналітик, експортується опис структур даних та накопичувачів даних, який за певними правилами формує опис сутностей та їх атрибутів.
І нарешті, **Rational Rose** — CASE-засіб фірми Rational Software Corporation (США) — призначений для автоматизації етапів аналізу та проектування ПЗ, а також для генерації кодів різними мовами та випуску проектної документації. Rational Rose використовує об'єднану методологію об'єктно-орієнтованого аналізу та проектування, засновану на підходах трьох провідних фахівців у цій галузі: Буча, Рамбо та Джекобсона. Розроблена ними універсальна нотація для моделювання об'єктів (UML – Unified Modeling Language) претендує на роль стандарту в галузі об'єктно-орієнтованого аналізу та проектування.
Конкретний варіант Rational Rose визначається мовою, якою генеруються коди програм (C++, Smalltalk, PowerBuilder, Ada, SQL Windows і ObjectPro). Основний варіант – Rational Rose/C++ – дозволяє розробляти проектну документацію у вигляді діаграм та 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.
Rational Rose працює на різних платформах: IBM PC (в середовищі Windows), Sun SPARC stations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 (AIX).

10.3. ВІЗУАЛЬНЕ МОДЕЛЮВАННЯ В RATIONAL ROSE

Вивчаючи вимоги до системи, ви берете за основу запити користувачів і перетворюєте їх на таку форму, яку ваша команда зможе зрозуміти і реалізувати. На основі цих вимог ви генеруєте код. Формально перетворюючи вимоги в код, ви гарантуєте їх відповідність один одному, а також можливість у будь-який момент повернутися від коду до вимог, що його породили. Цей процес називається моделюванням. Моделювання дозволяє простежити шлях від запитів користувачів до вимог моделі і потім коду і назад, не втрачаючи при цьому своїх напрацювань.
Візуальним моделюванням називають процес графічного представлення моделі з допомогою деякого стандартного набору графічних елементів. Наявність стандарту життєво необхідна реалізації однієї з переваг візуального моделювання — комунікації. Спілкування між користувачами, розробниками, аналітиками, тестувальниками, менеджерами та іншими учасниками проекту є основною метою графічного візуального моделювання.
Створені моделі надаються всім зацікавленим сторонам, які можуть отримати з них цінну інформацію. Наприклад, дивлячись на модель, користувачі візуалізують свою взаємодію з системою. Аналітики

побачать взаємодію між об'єктами моделі. Розробники зрозуміють, які об'єкти потрібно створити та що ці об'єкти мають робити. Тестувальники візуалізують взаємодію між об'єктами, що дозволить їм збудувати тести. Менеджери побачать як всю систему загалом, і взаємодія її частин. Нарешті, керівники інформаційної служби, дивлячись на високорівневі моделі, зрозуміють, як взаємодіють друг з одним системи у тому організації. Таким чином, візуальні моделі надають потужний інструмент, що дозволяє показати систему, що розробляється всім зацікавленим сторонам.

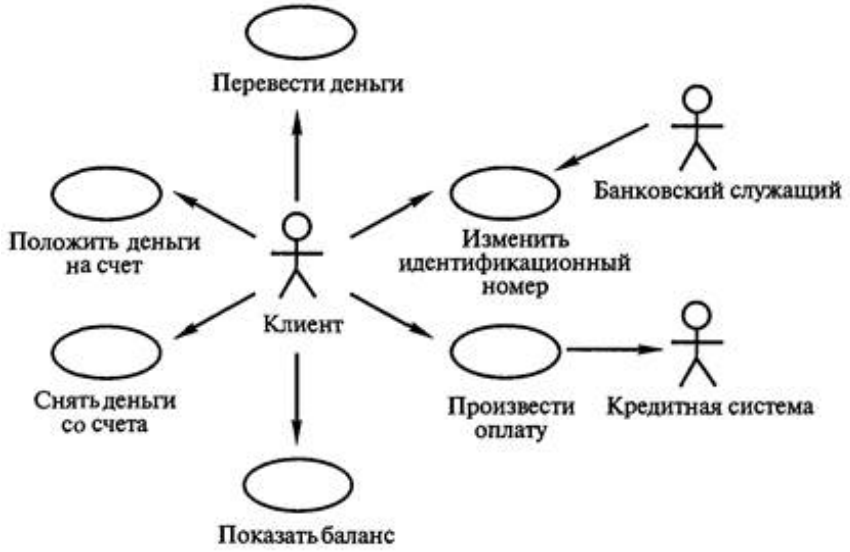
10.4. ДІАГРАМИ UML

10.4.1. Типи візуальних діаграм UML

UML дозволяє створювати кілька типів візуальних діаграм:
• діаграми варіантів використання;
• діаграми послідовності;
• кооперативні діаграми;
• діаграми класів;
• діаграми станів;
• діаграми компонентів;
• діаграми розміщення.
Діаграми ілюструють різні аспекти системи. Наприклад, кооперативна діаграма показує, як повинні взаємодіяти об'єкти, щоб реалізувати певну функціональність системи. Кожна діаграма має свою мету.

10.4.2. Діаграми варіантів використання

Діаграми варіантів використання відображають взаємодію між варіантами використання, що представляють функції системи, та дійовими особами, які представляють людей або системи, які отримують або передають інформацію до цієї системи. Приклад діаграми варіантів використання банківського автомата (АТМ) показаний на рис. 10.1.



Мал. 10.1. Діаграма варіантів використання

На діаграмі представлено взаємодію між варіантами використання та дійовими особами. Вона відображає вимоги до системи з погляду користувача. Таким чином, варіанти використання - це функції, що виконуються системою, а дійові особи - це зацікавлені особи по відношенню до створюваної системи. Діаграми показують, які дійові особи ініціюють варіанти використання. З них також видно, коли дійова особа одержує інформацію від варіанта використання. По суті, діаграма варіантів використання може ілюструвати вимоги до системи. У прикладі клієнт банку ініціює різні варіанти використання: «Зняти гроші з рахунку», «Перевести гроші», «Покласти гроші на рахунок»,

«Показати баланс», «Змінити ідентифікаційний номер», «Здійснити оплату». Банківський службовець може ініціювати варіант використання "Змінити ідентифікаційний номер". Від варіанта використання «Здійснити оплату» йде стрілка до Кредитної системи. Діючими особами можуть бути й зовнішні системи, у цьому випадку Кредитна система показана саме як дійова особа — вона є зовнішньою для системи АТМ. Стрілка, спрямована від варіанта використання до дійової особи, показує, що варіант використання надає деяку інформацію чинній особі. В даному випадку варіант використання «Здійснити оплату» надає Кредитній системі інформацію про оплату за кредитною карткою.
З діаграм варіантів використання можна отримати багато інформації про систему. Цей тип діаграм визначає загальну функціональність системи. Користувачі, менеджери проектів, аналітики, розробники, фахівці з контролю якості та всі, кого цікавить система загалом, можуть, вивчаючи діаграми варіантів використання, зрозуміти, що система має робити.

10.4.3. Діаграми послідовності

Діаграми послідовності відбивають потік подій, які у рамках варіанта використання. Наприклад, варіант використання «Зняти гроші» передбачає кілька можливих послідовностей: зняття грошей, спроба зняти гроші за відсутності їх достатньої кількості на рахунку, спроба зняти гроші за неправильним ідентифікаційним номером та деякі інші. Нормальний сценарій зняття $20 з рахунку (за відсутності таких проблем, як неправильний ідентифікаційний номер або нестача грошей на рахунку) показано на рис. 10.2.

*10.2.*Діаграма послідовності зняття клієнтом Джо $20 з рахунку

У верхній частині діаграми показані всі дійові особи та об'єкти, необхідні системі для виконання

варіанта використання «Зняти гроші». Стрілки відповідають повідомленням, що передаються між дійовою особою та об'єктом або між об'єктами для виконання потрібних функцій. Слід зазначити також, що у діаграмі послідовності показані саме об'єкти, а чи не класи. Класи є типами об'єктів. Об'єкти є конкретними; замість класу Клієнт на діаграмі послідовності подано конкретний клієнт Джо.
Варіант використання починається, коли клієнт вставляє свою картку в пристрій для читання – цей об'єкт показано у прямокутнику у верхній частині діаграми. Він зчитує номер картки, відкриває об'єкт "рахунок Джо" та ініціалізує екран АТМ. Екран запитує Джо його реєстраційний номер. Клієнт вводить число 1234. Екран перевіряє номер об'єкта «рахунок Джо» і виявляє, що він правильний. Потім екран надає Джо меню для вибору і той вибирає пункт «Зняти гроші». Екран запитує скільки він хоче зняти, і Джо вказує $20. Екран знімає гроші з рахунку. При цьому він ініціює серію процесів, які виконує об'єкт «рахунок Джо». У той самий час здійснюється перевірка, що у цьому рахунку лежать, по крайнього заходу, $20 і з рахунку віднімається необхідна сума. Потім касовий апарат отримує інструкцію "видати чек і $20 готівкою". Нарешті, той самий об'єкт «рахунок Джо» дає пристрою для читання карток інструкцію повернути картку.
Отже, дана діаграма послідовності ілюструє послідовність дій, що реалізують варіант використання "Зняти гроші з рахунку" на конкретному прикладі зняття клієнтом Джо $20. Дивлячись на цю діаграму, користувачі знайомляться зі специфікою своєї роботи. Аналітики бачать послідовність (потік) дій, розробники - об'єкти, які треба створити, та їх операції. Фахівці з контролю якості зрозуміють деталі процесу та зможуть розробити тести для їхньої перевірки. Таким чином, діаграми послідовності корисні для всіх учасників проекту.

10.4.4. Кооперативні діаграми

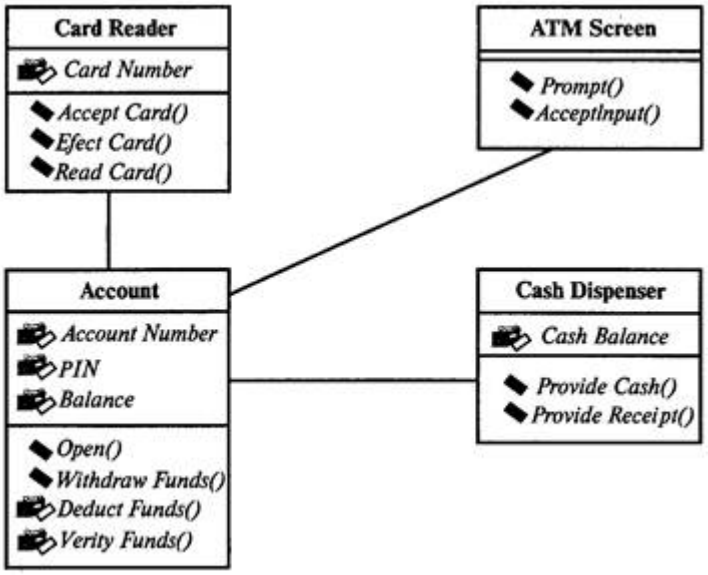
Кооперативні діаграми відображають ту саму інформацію, що і діаграми послідовності. Однак роблять вони це інакше і з іншими цілями. Показана на рис. 10.2 діаграма послідовності представлена на рис. 10.3 як кооперативної діаграми.
Як і раніше, об'єкти зображені у вигляді прямокутників, а дійові особи у вигляді фігур. Якщо діаграма послідовності показує взаємодію між дійовими особами та об'єктами у часі, то на кооперативній діаграмі зв'язок з часом відсутній. Так, можна бачити, що пристрій для читання картки видає "рахунку Джо" інструкцію відкритися, а "рахунок Джо" змушує цей пристрій повернути картку власнику. Безпосередньо об'єкти, що взаємодіють, з'єднані лініями. Якщо, наприклад, пристрій читання картки спілкується безпосередньо з екраном АТМ, між ними слід провести лінію. Відсутність лінії означає, що безпосереднє повідомлення між об'єктами відсутнє.

*Мал. 10.3.*Кооперативна діаграма, що описує процес зняття грошей з рахунку

Отже, на кооперативній діаграмі відображається та сама інформація, що і на діаграмі послідовності, але потрібна вона для інших цілей. Фахівці з контролю якості та архітектори системи зможуть зрозуміти розподіл між об'єктами. Припустимо, якась кооперативна діаграма нагадує зірку, де кілька об'єктів пов'язані з одним центральним об'єктом. Архітектор системи може зробити висновок, що система дуже залежить від центрального об'єкта, і необхідно перепроектувати її для більш рівномірного розподілу процесів. На діаграмі послідовності такий тип взаємодії було важко побачити.

10.4.5. Діаграми класів

Діаграми класів відбивають взаємодію між класами системи. Наприклад, "рахунок Джо" - це об'єкт. Типом такого об'єкта вважатимуться рахунок взагалі, т. е. «Рахунок» — це клас. Класи містять дані та поведінку (дії), що впливає на ці дані. Так, клас Рахунок містить ідентифікаційний номер клієнта та дії, що перевіряють. На діаграмі класів клас створюється для кожного типу об'єктів із діаграм послідовності або кооперативних діаграм. Діаграма класів для варіанта використання "Зняти гроші" показана на рис. 10.4.
На діаграмі показані зв'язки між класами, які реалізують варіант використання "Зняти гроші". У цьому процесі задіяно чотири класи: Card Reader (пристрій для читання карток), Account (рахунок), ATM (екран ATM) та Cash Dispenser (касовий апарат). Кожен клас на діаграмі класів зображується у вигляді прямокутника, розділеного на три частини. У першій частині вказується ім'я класу, у другій його атрибути. Атрибут - це деяка інформація, що характеризує клас. Наприклад, клас Account (рахунок) має три атрибути: Account Number (номер рахунку), PIN (ідентифікаційний номер) та Balance (баланс). В останній частині містяться операції класу, що відображають його поведінку (дії класу). Сполучні класи лінії показують взаємодію між класами.

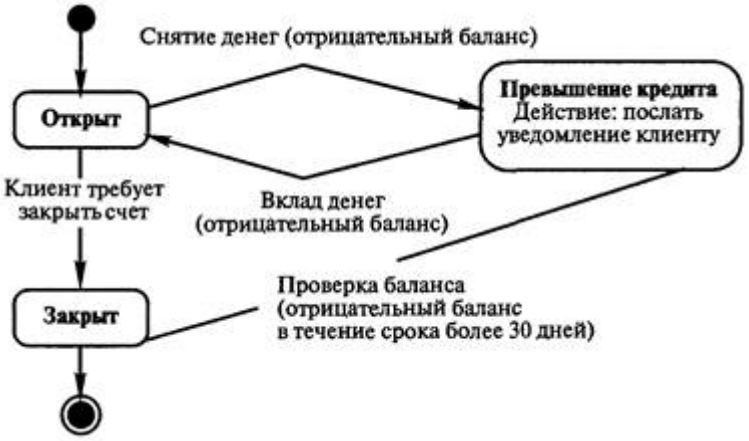


*Мал. 10.4.*Діаграма класів

Розробники використовують діаграми класів для створення класів. Такі інструменти, як Rose, генерують основу коду класів, яку програмісти заповнюють деталями обраною ними мовою. За допомогою цих діаграм аналітики можуть показати деталі системи, а архітектори зрозуміти її проект. Якщо, наприклад, якийсь клас несе надто велике функціональне навантаження, це буде видно на діаграмі класів, і архітектор зможе перерозподілити її між іншими класами. За допомогою діаграми можна виявити випадки, коли між сполученими класами не визначено жодних зв'язків. p align="justify"> Діаграми класів слід створювати, щоб показати взаємодіючі класи в кожному варіанті використання. Можна будувати також загальніші діаграми, що охоплюють усі системи чи підсистеми.

10.4.6. Діаграми станів

Діаграми станів призначені для моделювання різних станів, у яких може бути об'єкт. У той час, як діаграма класів показує статичну картину класів та їх зв'язків, діаграми станів застосовуються при описі динаміки поведінки системи.
Діаграми станів відображають поведінку об'єкта. Так, банківський рахунок може мати кілька різних статків. Він може бути відкритий, закритий або може бути перевищений кредит щодо нього. Поведінка рахунка змінюється залежно стану, у якому перебуває. На діаграмі станів показують цю інформацію. На рис. 10.5 наведено приклад діаграми станів для банківського рахунку.



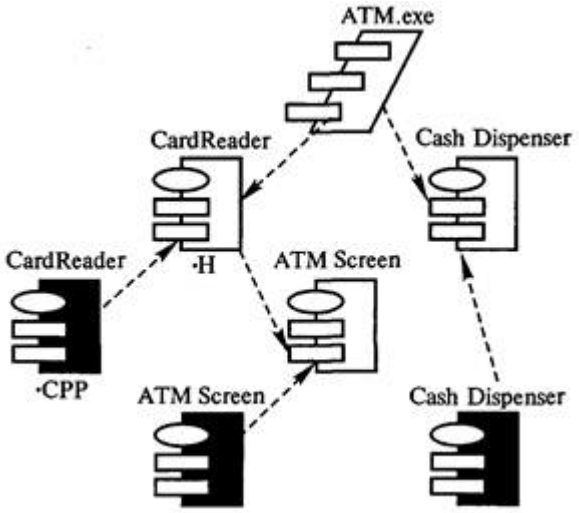
*Мал. 10.5.*Діаграма станів для класу Account

На цій діаграмі показані можливі стани рахунку, а також процес переходу рахунку з одного стану до іншого. Наприклад, якщо клієнт вимагає закрити відкритий рахунок, останній перетворюється на стан «Закритий». Вимога клієнта називається подією, саме події викликають перехід із одного стану в інший. Коли клієнт знімає гроші з відкритого рахунку, рахунок може перейти у стан "Перевищення кредиту". Це відбувається, тільки якщо баланс по рахунку менший за нуль, що відображено умовою [негативний баланс] на нашій діаграмі. Укладене в квадратні дужки умова визначає, коли може чи може статися перехід із одного стану до іншого.
На діаграмі є два спеціальні стани - початковий і кінцевий. Початковий стан виділяється чорною точкою: він відповідає стану об'єкта на момент створення. Кінцевий стан позначається чорною точкою у білому гуртку: він відповідає стану об'єкта безпосередньо перед його знищенням. На діаграмі станів може бути один і лише один початковий стан. У той же час може бути стільки кінцевих станів, скільки вам потрібно, або їх може не бути взагалі.
Коли об'єкт перебуває у певному стані, можуть виконуватися ті чи інші процеси. У прикладі при перевищенні кредиту клієнту надсилається відповідне повідомлення. Процеси, що відбуваються, коли об'єкт перебуває у певному стані, називаються діями.
Діаграми станів не потрібно створювати для кожного класу, вони використовуються тільки в дуже складних випадках. Якщо об'єкт класу може існувати в кількох станах і в кожному з них поводиться по-різному, для нього, ймовірно, буде потрібно таку діаграму. Однак у багатьох проектах вони взагалі не використовуються. Якщо ж діаграми станів таки були побудовані, розробники можуть застосовувати їх під час створення класів.
Діаграми станів необхідні в основному для документування.

10.4.7. Діаграми компонент

Діаграми компонент показують, як виглядає модель фізично. На ній зображуються компоненти програмного забезпечення вашої системи та зв'язку між ними. При цьому виділяють два типи компонентів: виконувані компоненти та бібліотеки коду.
На рис. 10.6 зображено одну з діаграм компонент для системи ATM. На цій діаграмі показано компоненти клієнта системи ATM. У разі команда розробників вирішила будувати систему з допомогою мови C++. Кожен клас має свій власний заголовний файл і файл з розширенням .CPP, отже кожен клас перетворюється на власні компоненти на діаграмі. Виділена темна компонента називається

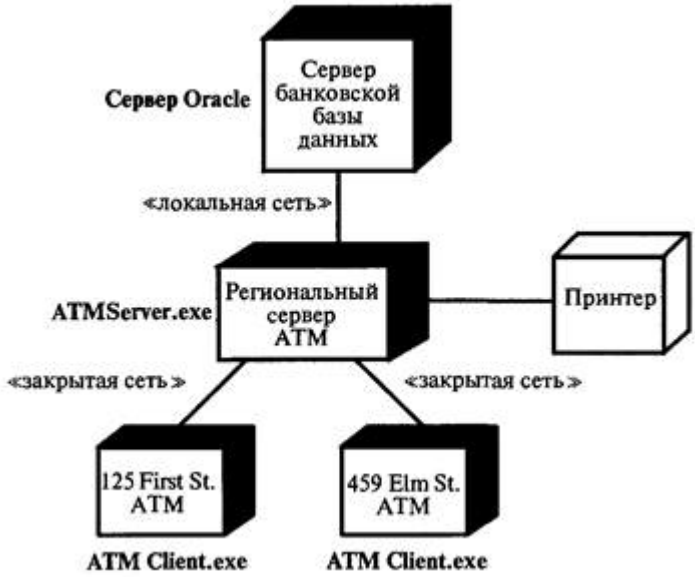
специфікацією пакета і відповідає файлу тіла класу ATM мовою C++ (файл із розширенням .CPP). Невиділена компонента називається специфікацією пакета, але відповідає заголовному файлу класу мови C++ (файл з розширенням .H). Компонента ATM.exe є специфікацією завдання та представляє потік обробки інформації. У разі потік обробки — це виконувана програма.
Компоненти з'єднані штриховою лінією, що відображає залежність між ними. Система може мати кілька діаграм компонентів залежно від кількості підсистем або виконуваних файлів. p align="justify"> Кожна підсистема є пакетом компонент.
Діаграми компонентів застосовуються тими учасниками проекту, хто відповідає за компіляцію системи. Діаграма компонент дає уявлення про те, в якому порядку треба компілювати компоненти, а також які компоненти, що виконуються, будуть створені системою. Діаграма показує відповідність класів реалізованим компонентам. Отже, вона потрібна там, де починається створення коду.



*Мал. 10.6.*Діаграма компонент

10.4.8. Діаграми розміщення

Діаграми розміщення показують фізичне розташування різних компонентів системи в мережі. У прикладі система ATM складається з великої кількості підсистем, виконуваних на окремих фізичних пристроях чи вузлах. Діаграма розміщення системи ATM представлена на рис. 10.7.
З цієї діаграми можна дізнатися про фізичне розміщення системи. Клієнтські програми ATM працюватимуть у кількох місцях різних сайтах. Через закриті мережі буде здійснюватись повідомлення клієнтів з регіональним сервером ATM. На ньому працюватиме програмне забезпечення сервера ATM. У свою чергу, за допомогою локальної мережі регіональний сервер взаємодіятиме з сервером банківської бази даних, який працює під управлінням Oracle. Нарешті, з регіональним ATM сервером з'єднаний принтер.
Отже, ця діаграма показує фізичне розташування системи. Наприклад, наша система ATM відповідає трирівневій архітектурі, коли на першому рівні розміщується база даних, на другому регіональний сервер, а на третьому клієнт.



*10.7.*Діаграма розміщення

Діаграма розміщення використовується менеджером проекту, користувачами, архітектором системи та експлуатаційним персоналом для з'ясування фізичного розміщення системи та розташування її окремих підсистем. Менеджер проекту пояснить користувачам, як виглядатиме готовий продукт. Експлуатаційний персонал зможе планувати роботу із встановлення системи.

10.5. ВІЗУАЛЬНЕ МОДЕЛЮВАННЯ І ПРОЦЕС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

10.5.1. Переваги та недоліки типів процесу розробки

Програмне забезпечення може бути створене різними способами. Існує кілька різних типів процесу розробки, які можна використовувати у проекті: від «водоспаду» до объектно-ориентированного підходу. У кожного є свої переваги та недоліки. Тут не вказується, який саме процес проектування необхідно застосовувати розробникам своєї роботи, а представляємо лише короткий опис процесу, пов'язаного з візуальним моделюванням.
Довгий час програмне забезпечення розробляли, дотримуючись так званої моделі «водоспаду». Відповідно до неї необхідно було спочатку проаналізувати вимоги до майбутньої системи, спроектувати, створити та протестувати систему, а потім встановити її у користувачів. Як видно з назви, рух у зворотний бік по цьому ланцюжку було неможливим — вода не потече вгору. Цей метод був офіційною методологією, що застосовувалась у тисячах проектів, але в чистому вигляді, як його прийнято розуміти, він не використовувався жодного разу. Одним із головних недоліків моделі «водоспаду» є неможливість повернення назад на пройдені етапи. На початку проекту, наступного такої моделі, перед розробниками стоїть завдання, що бентежить — повністю визначити всі вимоги до системи. Для цього необхідно ретельно та всебічно обговорити з користувачами та дослідити бізнес-процеси. Користувачі повинні погодитися з усім тим, що з'ясовується в ході такого обстеження, хоча вони можуть до кінця не ознайомлюватися з його результатами. У такий спосіб на стадії аналізу при деякому везенні вдається зібрати близько 80% вимог до системи.
Потім розпочинається етап проектування. Необхідно сісти та визначити архітектуру майбутньої системи. Ми з'ясовуємо, де будуть встановлені програми та яка апаратура потрібна для досягнення прийнятної продуктивності. На даному етапі можуть виявитися нові проблеми, їх необхідно знову обговорювати з користувачами, що виявиться у появі нових вимог до системи. Отже, доводиться повертатись до аналізу.
Після кількох таких кіл нарешті настає етап написання програмного коду. На цьому етапі виявляється, що деякі з ухвалених раніше рішень неможливо здійснити. Доводиться повертатися до проектування та

переглядати ці рішення. Після завершення кодування настає етап тестування, у якому з'ясовується, що вимоги були досить деталізовані та його реалізація некоректна. Потрібно повертатись назад, на етап аналізу, та переглядати ці вимоги.
Нарешті, система готова та поставлена користувачам. Оскільки минуло вже багато часу і бізнес, ймовірно, вже встиг змінитися, користувачі сприймають її без великого ентузіазму, відповідаючи приблизно так: «Так, це те, що я просив, але не те, що хочу!» Ця магічна фраза, заклинання разом зістарить всю команду розробників щонайменше на 10 років!
Чи проблема полягає в тому, що правила бізнесу змінюються занадто швидко? Можливо, користувачі не можуть пояснити, чого вони хочуть чи не розуміють команду розробників? Чи сама команда не дотримується певного процесу? Відповіді на ці питання: так, та й ні. Бізнес змінюється дуже швидко, і професіонали-програмісти повинні встигати за цим. Користувачі не завжди можуть сказати, чого вони хочуть, оскільки їхня робота стала для них другою натурою. Запитувати про це банківського клерка, що прослужив 30 років, — приблизно те саме, що питати, як він дихає. Робота стала для нього настільки звичною, що її важко описати. Ще одна проблема полягає в тому, що користувачі не завжди розуміють команду розробників. Команда показує їм графіки, випускає томи тексту, що описує вимоги до системи, але користувачі не завжди розуміють, що їм дають. Чи є спосіб оминути цю проблему? Так, тут допоможе візуальне моделювання. І нарешті, команда розробників точно дотримується процесу — методу «водоспаду». На жаль, планування та реалізація методу – дві різні речі.
Отже, одна з проблем полягає в тому, що розробники використовували метод «водоспаду», який полягає в акуратному та послідовному проходженні через усі етапи проекту, але їм доводилося повертатися на пройдені етапи. Чи це відбувається через погане планування? Мабуть, ні. Розробка програмного забезпечення - складний процес, і його поетапне, акуратне виконання не завжди можливе. Якщо ж ігнорувати необхідність повернення, система буде містити дефекти проектування, і деякі вимоги будуть втрачені, можливі і серйозніші наслідки. Минули роки, допоки ми не навчилися заздалегідь планувати повернення на пройдені етапи.
Таким чином, ми дійшли ітеративної розробки. Ця назва означає лише, що ми збираємося повторювати ті самі етапи знову і знову. В об'єктно-орієнтованому процесі потрібно по багато разів невеликими кроками проходити етапи аналізу, проектування, розробки, тестування та встановлення системи. Неможливо виявити всі вимоги ранніх етапах проектування. Ми враховуємо появу нових вимог у процесі розробки, плануючи проект у кілька ітерацій. У межах такої концепції проект можна як послідовність невеликих «водопадиков». Кожен із них досить великий, щоб означати завершення будь-якого важливого етапу проекту, але малий, щоб мінімізувати необхідність повернення назад. Ми проходимо чотири фази (етапу) проекту: початкова фаза, уточнення, конструювання та введення в дію.
Початкова фаза- Це початок проекту. Ми збираємо інформацію та розробляємо базові концепції. Наприкінці цієї фази приймається рішення продовжувати (чи продовжувати) проект.
У фазі уточнення деталізуються варіанти використання та приймаються архітектурні рішення. Уточнення включає деякий аналіз, проектування, кодування та планування тестів.
У фазі конструювання розробляється переважна більшість коду.
Введення в дію— це завершальне компонування системи та встановлення її у користувачів. Далі розглянемо, що означає кожна з цих фаз об'єктно-орієнтованому проекті.

10.5.2. Початкова фаза

Початкова фаза— це початок роботи над проектом, коли хтось каже: «А добре, щоб система робила...» Потім хтось ще вивчає ідею, і менеджер запитує, скільки часу вимагатиме її реалізація, скільки це коштуватиме і наскільки вона здійсненна. Початкова фаза таки полягає у тому, щоб знайти відповіді на ці питання. Ми досліджуємо властивості системи на високому рівні та документуємо їх. Визначаємо дійових осіб та варіанти використання, але не заглиблюємося в деталі варіантів використання, обмежуючись однією або двома пропозиціями. Готуємо також оцінки для найвищого керівництва.
Отже, застосовуючи Rose для підтримки нашого проекту, створюємо дійових осіб та варіанти використання, а також будуємо діаграми варіантів використання. Початкова фаза завершується, коли це дослідження закінчено і для роботи над проектом виділені необхідні ресурси.
Початкова фаза проекту переважно послідовна і ітеративна. На відміну від неї інші фази повторюються кілька разів у процесі роботи над проектом. Оскільки проект може бути розпочато лише один раз,

початкова фаза також виконується лише одного разу, тому в початковій фазі має бути вирішене ще одне завдання – розробка плану ітерацій. Це план, який описує, які варіанти використання, на яких ітераціях мають бути реалізовані. Якщо, наприклад, у початковій фазі виявлено 10 варіантів використання, можна створити наступний план:
Ітерація 1 Варіанти Використання 1, 5, 6
Ітерація 2 Варіанти Використання 7, 9
Ітерація 3 Варіанти Використання 2, 4, 8
Ітерація 4 Варіанти Використання 3, 10
План визначає, які варіанти використання треба реалізувати насамперед. Побудова плану вимагає розгляду залежностей між варіантами використання. Якщо для того, щоб міг працювати Варіант Використання 5, необхідна реалізація Варіанта Використання 3, то описаний вище план нездійсненний, оскільки Варіант Використання 3 реалізується на четвертій ітерації значно пізніше Варіанта Використання 5 з першої ітерації. Такий план слід переглянути, щоб врахувати всі залежності.

10.5.3. Використання Rose у початковій фазі

Деякі завдання початкової фази включають визначення варіантів використання і дійових осіб. Rose можна застосовувати для документування цих варіантів використання та дійових осіб, а також для створення діаграм, що показують зв'язок між ними. Отримані діаграми варіантів використання можна показати користувачам, щоб переконатися, що вони дають повне уявлення про властивості системи.
У фазі уточнення проекту виконуються деяке планування, аналіз та проектування архітектури. Наслідуючи план ітерації, уточнення проводиться для кожного варіанту використання в поточній ітерації. Уточнення включає такі аспекти проекту, як кодування прототипів, розробка тестів і прийняття рішень по проекту.
Основне завдання фази уточнення – деталізація варіантів використання. Вимоги низького рівня, що пред'являються до варіантів використання, передбачають опис потоку обробки даних усередині них, виявлення дійових осіб, розробку діаграм Взаємодії для графічного відображення потоку обробки даних, а також визначення всіх переходів станів, які можуть мати місце в рамках варіанту використання. p align="justify"> З вимог, визначених у формі деталізованих варіантів використання, складається документ під назвою «Специфікація вимог до програмного забезпечення».
У фазі уточнення виконуються такі завдання, як уточнення попередніх оцінок, вивчення моделі варіантів використання з погляду якості проектованої системи, аналіз ризиків. Можна уточнити модель варіантів використання, а також розробити діаграми послідовності та кооперативні діаграми для графічного представлення потоку обробки даних. Крім того, у цій фазі проектуються діаграми класів, що описують об'єкти, які необхідно створити.
Фаза уточнення завершується, коли варіанти використання повністю деталізовані та схвалені користувачами, прототипи завершені настільки, щоб зменшити ризики, розроблені діаграми класів. Іншими словами, ця фаза пройдена, коли система спроектована, розглянута та готова для передачі розробникам.
Фаза уточнення пропонує кілька можливостей для застосування Rational Rose. Оскільки уточнення – це деталізація вимог до системи, модель варіантів використання може вимагати оновлення. Діаграми послідовності та кооперативні діаграми допомагають проілюструвати потік обробки даних при його деталізації. З їхньою допомогою можна також спроектувати необхідні для системи об'єкти. Уточнення передбачає підготовку проекту системи передачі розробникам, які розпочнуть її конструювання. У середовищі Rose може бути виконано шляхом створення діаграм класів і діаграм станів.
Конструювання- Процес розробки та тестування програмного забезпечення. Як і у разі уточнення, ця фаза виконується для кожного набору варіантів використання кожної ітерації. Завдання конструювання включають визначення всіх вимог, розробку і тестування програмного забезпечення (ПО). Так як ПЗ повністю проектується у фазі уточнення, конструювання не передбачає великої кількості рішень щодо проекту, що дозволяє команді працювати паралельно. Це означає, що різні групи програмістів можуть одночасно працювати над різними об'єктами, знаючи, що після завершення фази система «зійдеться». У фазі уточнення ми проектуємо об'єкти системи та їхню взаємодію. Конструювання лише запускає проект у дію, а нових рішень щодо нього, здатних змінити цю взаємодію, не приймається.

Ще однією перевагою такого підходу до моделювання системи є те, що середовище Rational Rose здатне генерувати «скелетний код» системи. Для використання цієї можливості слід розробити компоненти та діаграму компонентів на ранньому етапі конструювання. Генерацію коду можна розпочати відразу після створення компонентів та нанесення на діаграму залежностей між ними. В результаті буде автоматично побудований код, який можна створити, ґрунтуючись на проекті системи. Це не означає, що за допомогою Rose можна отримати будь-який код, що реалізує бізнес-логіку програм. Результат сильно залежить від мови програмування, але в загальному випадку передбачає визначення класів, атрибутів, областей дії. Це дозволяє заощадити час, тому що написання коду вручну — досить копітка та стомлююча робота. Отримавши код програмісти можуть сконцентруватися на специфічних аспектах, пов'язаних з бізнес-логікою. Ще одна група розробників має виконати експертну оцінку коду, щоб переконатися в його функціональності та відповідності стандартам та угодам щодо проекту. Потім об'єкти повинні бути оцінені якістю. Якщо у фазі конструювання були додані нові атрибути або функції або якщо були змінені взаємодії між об'єктами, код слід перетворити на модель Rose за допомогою зворотного перетворення.
Конструювання можна вважати завершеним, коли програмне забезпечення готове та протестоване. Важливо переконатися в адекватності моделі та програмного забезпечення. Модель буде надзвичайно корисною у процесі супроводу ПЗ.
У фазі конструювання пишеться більшість коду проекту. Rose дозволяє створити компоненти відповідно до проектування об'єктів. Щоб показати залежність між компонентами на етапі компіляції, створюються діаграми компонентів. Після вибору мови програмування можна здійснити генерацію скелетного коду кожного компонента. Після завершення роботи над кодом модель можна привести у відповідність з ним за допомогою зворотного проектування.
p align="justify"> Фаза введення в дію настає, коли готовий програмний продукт передають користувачам. Завдання у цій фазі передбачають завершення роботи над фінальною версією продукту, завершення приймального тестування, завершення складання документації та підготовку до навчання користувачів. Щоб відобразити останні внесені зміни, слід оновити специфікацію вимог до програмного забезпечення, діаграми варіантів використання, класів, компонентів та розміщення. Важливо, щоб ваші моделі були синхронізовані з готовим продуктом, оскільки вони використовуватимуться під час його супроводу. Крім того, моделі будуть неоціненними при внесенні удосконалень у створену систему вже за кілька місяців після завершення проекту. У фазі введення в дію Rational Rose не така корисна, як в інших фазах. У цей момент програма вже створена. Rose призначена для надання допомоги при моделюванні та розробці програмного забезпечення і навіть при плануванні його розміщення. Однак Rose не є інструментом тестування і не здатна допомогти у плануванні тестів або процедур, пов'язаних із розміщенням програмного забезпечення. Для цього створені інші продукти. Отже, у фазі введення в дію Rose застосовується передусім для оновлення моделей після завершення роботи над програмним продуктом.

10.6. РОБОТА НАД ПРОЕКТОМ У СЕРЕДОВИЩІ RATIONAL ROSE

З усіх розглянутих видів канонічних діаграм серед Rational Rose 98/98i не підтримується лише діаграма діяльності.
У ході роботи над діаграмами проекту є можливість видалення та додавання відповідних графічних елементів, встановлення відносин між цими елементами, їх специфікації та документування.
Загальна послідовність роботи над проектом аналогічна послідовності розгляду канонічних діаграм у книзі.
Однією з найпотужніших властивостей середовища Rational Rose є можливість генерації програмного коду після побудови та перевірки моделей. Загальна послідовність дій, які необхідно виконати для цього, складається із шести етапів:
- Перевірки моделі незалежно від вибору мови генерації коду;
- Створення компонентів для реалізації класів;
- Відображення класів на компоненти;
- Налаштування властивостей генерації програмного коду;
- вибору класу, компонента чи пакета;
- генерації програмного коду.

ВИСНОВКИ

• CASE-кошти дозволяють в автоматизованому режимі реалізувати проектні моделі.
• Реалізовані проектні моделі повинні бути повними, відображати як функціональні, так і інформаційні аспекти автоматизованих систем, що проектуються.
• CASE-засоби включають набір інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки та супроводу програмного проекту та розробляти програми відповідно до інформаційних потреб користувачів.
• Більшість існуючих CASE-засобів засновані на методологіях структурного (в основному) або об'єктно-орієнтованого аналізу та проектування, що використовують специфікації у вигляді діаграм або текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поведінки системи та архітектури програмних засобів.
• Передові засоби SESE здатні не тільки складати специфікації, але і їх перевіряти, а також генерувати вихідний код програм.
• CASE-засоби виробляються безліччю виробників і лише найвдаліші з них проходять перевірку практикою.
• CASE-засіб Rational Rose фірми «Software Corporation» (США) призначений для автоматизації етапів аналізу та проектування ПЗ, а також для генерації кодів різними мовами та випуску проектної документації. Rational Rose використовує об'єднану методологію об'єктно-орієнтованого аналізу та проектування, засновану на підходах трьох провідних фахівців у цій галузі: Буча, Рамбо та Джекобсона.
• Мова UML CASE-засоби Rational Rose дозволяє створювати кілька типів візуальних діаграм:
- Діаграми варіантів використання;
- Діаграми послідовності;
- Кооперативні діаграми;
- Діаграми класів;
- Діаграми станів;
- Діаграми компонент;
- Діаграми розміщення.

Контрольні питання

1. Що таке CASE-кошти?
2. Навіщо необхідні CASE-кошти?
3. У чому полягає суть візуального моделювання?
4. Що відображає діаграми варіантів використання?
5. Що відображають діаграми послідовності?
6. Що відображають кооперативні діаграми?
7. Що відображають діаграми класів?
8. Що відображає діаграми станів?
9. Що відображають діаграми компонентів?
10. Що відображають діаграми розміщення?
11. У чому полягає суть моделі розробки програмного забезпечення «водоспад», її особливості та недоліки?
12. Викладіть кроки методики розробки програм із використанням Rational Rose.