**Лекція 9. Прикладні та теоретичні методи програмування**

**Поняття прикладних і теоретичних методів програмування.**

Кожна парадигма програмування характеризується наявністю в ній метода і зв’язком з моделлю ЖЦ. Парадигми програмування об’єднують загальні положення з проектування рограмного продукту (ПП). Користувач може вибирати ту або іншу парадигму програмування з позицій зручності застосування для задач у предметній обласиі (ПрО) і виготовлення конкретного ПП.

Розрізняють прикладні і теоретичні методи, які з’явилися у різний час від моменту появи програмної інженерії і мають свою специфіку й сферу застосування. Так, структурний метод виник багато років тому, він добре відпрацьований і вдосконалений для індустріального виготовлення ПП. Широкого застосування набули методи модульного і компонентного програмування. Вони базуються на концепції повторного використання компонентів. Саме ці види програмування започаткували індустрію виготовлення ПП з готових компонентів. Поява нових методів стимулюється досягненнями загальнонаукових дисциплін (математики, логіки, теорії алгоритмів тощо) і практичними задачами.

*Теоретичні методи програмування* дозволяють створювати програмні системи в абстрактному вигляді, включаючи концептуальні, інформаційні, структурні моделі без урахування особливостей середовищ, в яких вони реалізуються. Ці моделі доводяться до стану кінцевого продукту шляхом використання відповідних мов програмування та їх перетворення.

Знаходять застосування формальні й теоретичні методи програмування (алгебраїчний, алгебро-алгоритмічний, композиційний й ін.), які ґрунтуються на математичних і логіко-алгоритмічних підходах до абстрактного створення ПП.

**Прикладне (систематичне) програмування**

До методів систематичного програмування відносять такі методи:

– структурний;

– об’єктно-орієнтований;

– UML-метод;

– компонентний;

– аспектно-орієнтований;

– генерувальний;

– сервісний;

– агентний й ін.

**Структурне програмування**

*Сутність структурного підходу* до розробки ПС полягає в декомпозиції (розподілі) системи на функції, що підлягають автоматизації, які у свою чергу, діляться на підфункції й задачі. Процес декомпозиції триває до визначення конкретних процедур. При цьому система, що автоматизується, зберігає цілісне подання, у якому всі складові компоненти взаємозалежні.

*Основу структурного програмування* становлять:

– розподіл системи на множину незалежних задач, доступних для розуміння і розв’язання;

– впорядкування й організація складових частин проблеми в ієрархічні деревоподібної структури з додаванням нових деталей на кожному рівні.

До *головних принципів* належать:

– абстрагування, тобто відокремлення істотних аспектів системи й нехтування несуттєвими;

– формалізація, тобто загальне методологічне вирішення проблеми;

– обґрунтування й узгодження елементів системи і перевірка їх на несуперечність;

– утворення ієрархічної структури даних.

При структурному аналізі застосовуються три найпоширеніші моделі структурного проектування програмних систем (ПС):

* SADT (Structured Analysis and Design Technique) – метод структурного аналізу й техніка проектування моделі системи за допомогою функціональних діаграм;
* SSADM (Structured Systems Analysis and Design Method) – метод структурного аналізу й проектування систем;
* IDEF (Integrated Definition Functions) – метод визначення функціональної моделі, IDEF1 – інформаційної моделі, IDEF2 – динамічної моделі й ін.

**Метод функціонального моделювання SADT**запропоновано Д.Россом і покладено в основу методології IDEF0 (Icam DEFinition), що є головною частиною програми ICAM (Інтеграція комп'ютерних і промислових технологій), проведеної з ініціативи ВПС США.

На стадії проектування моделі системи зображаються у вигляді діаграм або екранних форм і відображають структуру або архітектуру системи, а також схеми програм.

SADT – це сукупність правил і процедур, призначених для побудови функціональної моделі предметної області, яка відображає функціональну структуру, функції і дії, а також зв'язки між ними.

Метод SADT базується на наступних концепціях:

– графічне зображення структури з поданням функцій блоками, а інтерфейсів дугами, що, відповідно, входять у блок і виходять з нього (рис. 1);

**** Рис. 1. Структура моделі

– блоків може бути від 3 до 6 на кожному рівні декомпозиції;

– взаємодія блоків описується обмеженнями, які визначають умови керування й виконання функцій;

– унікальність позначок і найменувань;

– незалежність функціональної моделі від організаційної структури колективу розробників.

Метод SADT застосовується при моделюванні широкого кола систем, для яких визначаються вимоги й функції, а потім проводиться їхня реалізація. Засоби SADT можуть застосовуватися при аналізі функцій у діючій ПС, а також при визначенні способів їхньої реалізації.

**Метод SSADM** базується на таких структурах: послідовність, вибір й ітерація. Об’єкт моделювання задається відповідними структурними діаграмами, які відображають послідовність операторів, вибір елементів із групи й циклічне виконання операторів за цими елементами. Загальна діаграма системи згідно з цим методом має ієрархічну структуру і містить у собі: список компонентів модельованого об'єкта; ідентифіковані групи вибраних і повторюваних компонентів; послідовність використовуваних компонентів.

Таке програмування передбачає наявність моделі ЖЦ із послідовними процесами розроблення програмного проекту, починаючи з аналізу і формування вимог для ПрО

До процесів ЖЦ належать:

– стратегічне проектування та вивчення можливості виконання проекту;

– детальне дослідження предметної області, що містить у собі аналіз і специфікацію вимог;

– фізичне проектування структур даних відповідно до вибраної структури БД (реляційної, об’єктно-орієнтованої й ін.) та конструювання окремих компонентів, їх

тестування і тестування системи в цілому;

– виготовлення продукту і документації з нього для замовника.

*Детальне дослідження предметної област*і проводиться для того, щоб вивчити її особливості, розглянути потреби й пропозиції замовника, провести аналіз вимог з різних документів, специфікувати їх і погодити із замовником.

*Мета стратегічного проектування* – визначення сфери дії проекту, аналіз інформаційних потоків, формування загальної архітектури системи, визначення витрат на розробку і підтвердження можливості подальшої реалізації проекту.

*Результат* – це специфікація вимог, що застосовується при розроблені логічної структури системи.

*Логічне проектування* – це визначення функцій, діалогу, методу побудови і відновлення БД. У логічній моделі відображаються вхідні й вихідні дані, проходження запитів і встановлення взаємозв'язків між сутностями та подіями.

*Фізичне проектування* – це визначення типу СКБД і подання даних у ній з урахуванням специфікації логічної моделі даних, обмежень на пам'ять і час обробки, а також визначення механізмів доступу, розміру логічної БД, зв'язків між елементами системи. Фізична специфікація містить у собі:

– специфікацію функцій і схеми реалізації компонентів функцій,

– опис процедурних і непроцедурних компонентів й інтерфейсів,

– визначення логічних і фізичних груп даних з урахуванням обмежень устаткування на розробку й стандарти розробки,

– визначення груп подій, які обробляються як єдине ціле з видачею повідомлень про завершення обробки й ін.

Процеси, які виконуються у SSADM, пов'язані з роботами, що керують потоками інформації трьох типів: потік робіт; санкціоновані потоки за контролем або керуванням; звіти про хід розроблення.

*Конструювання* – це побудова конструкцій і елементів системи, їхнє тестування на наборах даних, які підбираються на ранніх процесах ЖЦ розробки системи.

Життєвий цикл містить у собі процес керування і контролю, який базується на сітковому графіку, що враховує роботи з розробки системи, витрати і строки. У графіку містяться роботи й взаємозв'язки між ними і їхніми виконавцями, а також проектні документи, які розроблюються виконавцями. Результати кожного з процесів ЖЦ контролюються і передаються на наступний етап у вигляді, зручному для подальшої реалізації іншими виконавцями.

**Метод IDEF1** базується на концепції ER-моделювання і призначений для побудови інформаційної моделі подібно до реляційної моделі. Основна особливість полягає в тому, що кожен екземпляр сутності може бути однозначно ідентифікований без визначення відношення з іншими сутностями. Якщо ідентифікація екземпляра сутності залежить від його відношення до іншої сутності, то сутність є залежною.

Обмеження на множинність зв’язку можуть означати, що для кожного екземпляра сутності-батька існує:

– нуль, один або більше пов'язаних з ним екземплярів сутності-нащадка;

– не менше ніж один або не більше ніж один пов'язаний з ним екземпляр сутності-нащадка;

– зв'язок з деяким фіксованим числом екземплярів сутності-нащадка.

Засобами IDEF1 проводиться збирання і вивчення різних сфер діяльності підприємства, визначення потреб в інформаційному менеджменті, а також:

– інформації й структури потоків, що властиві діяльності підприємства;

– правил і законів руху інформаційних потоків і принципів керування ними;

– взаємозв'язків між існуючими інформаційними потоками на підприємстві;

–проблем, що виникають при неякісному інформаційному менеджменті і потребують усунення.

Одна з особливостей даної методології – забезпечення структурованого процесу аналізу інформаційних потоків підприємства і можливості зміни неповної й неточної структури інформації на процесі моделювання інформаційної структури підприємства.

**Об’єктно-орієнтоване програмування**

*Об’єктно-орієнтований метод* програмування визначає стратегію побудови об’єктної системи, згідно з якою розробники системи мають мислити в термінах об'єктів, а не функцій. Об'єкт – це певна сутність, що перебуває в різних станах і має певний набір операцій. Операції, що пов’язані з об’єктом, надають іншим об’єктам послуги (сервіси) для виконання певних функцій, а їх стан залежить від значень атрибутів. Об'єкти створюються відповідно до визначення класу об’єктів, в якому описуються всі їхні атрибути й операції.

Модель об’єктно-орієнтованої програмної системи можна розглядати як набір взаємодіючих об'єктів, що мають власний стан і набір операцій, які впливають на стан інших об'єктів. Об'єкти приховують інформацію про значення станів, операцій і обмежують доступ до них.

*Операції над об'єктами*:

– введення, збереження, видалення об’єктів тощо, тобто це операції ЖЦ об’єктів;

– операції взаємодії об'єктів шляхом викликів методів об'єктів, визначених на множині вхідних і вихідних інтерфейсів.

*Інтерфейс* називається *вхідним*, якщо об'єкт за його допомогою одержує певний сервіс, і *вихідним,* якщо об'єкт через нього надає цей сервіс.

*Основною операцією об'єкта* є операція запиту*,* де визначені дія і список параметрів, заданих клієнтом для звернення до обслуговуючого об'єкта і отримання від нього результату. Запит виконується, якщо типи параметрів або результатів операції з ім’ям *w* відповідають множині вхідних і вихідних інтерфейсів. Під час виконання операції аргументи зв’язуються з формальними параметрами операції. На основі виконання операцій об'єкт здатний перебувати в різних станах. Кожний стан визначається набором атрибутів об'єкта, що задаються, і операцій, особливістю яких є поліморфізм. Операції об'єкта дозволяють одержати сервіс у об'єкта шляхом виконання певних обчислень, а потім отриманий результат надати іншим об'єктам.

Зміна реалізації якого-небудь об'єкта або додавання йому нових функцій не впливає на інші об'єкти системи. Чітка відповідність між реальними об'єктами (наприклад, апаратними засобами) і керуючими об'єктами ПС полегшує розуміння і реалізацію системи за її моделлю і об'єктами.

Об’єктно-орієнтована модель програмної системи створюється на таких процесах ЖЦ (рис. 2):

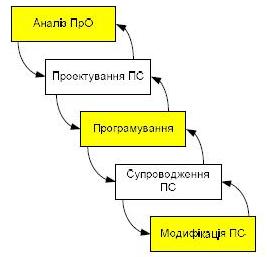


Рис. 2. ЖЦ розробки моделі системи у середовищі ООП

Етапам відповідають такі процеси:

*– аналіз –* створення об’єктної моделі (ОМ) ПрО, у якій об'єкти відбивають її реальні сутності і операції над ними;

*– проектування –* уточнення ОМ з урахуванням опису вимог для реалізації конкретних задач системи;

*– програмування –* реалізація ОМ засобами мов програмування С++, Java та ін.;

*– супроводження –* використання й розвиток системи шляхом внесення змін у об'єкти або в методи;

*– модифікація ПС* в процесі її супроводження шляхом додавання нових функціональних можливостей, інтерфейсів і операцій.

Наведені процеси можуть виконуватися ітераційно один за одним і з поверненням до попереднього процесу. На кожному процесі може застосовуватися та сама система нотацій.

Перехід до наступного процесу зумовлює вдосконалення результатів попереднього процесу шляхом більш детальної розробки раніше визначених класів об'єктів і додавання нових класів.

Результат процесу аналізу ЖЦ – модель ПрО й набір інших моделей (модель архітектури, модель оточення й використання). Моделі відображають зв'язки між об'єктами, їхні стани та набір операцій для динамічної зміни стану інших об'єктів, а також їх відношення із навколишнім середовищем.

Існує два типи моделей системної архітектури:

– *статична модель* для опису статичної структури системи в термінах класів об'єктів і відношень між ними (узагальнення, розширення, використання, успадкування);

– *динамічна модель* для опису динамічної структури системи і взаємодії між об'єктами (але не класами об’єктів) під час роботи системи.

Об'єкти інкапсулюють інформацію про свій стан і обмежують доступ до своїх атрибутів. Моделі оточення й використання системи – це дві моделі, що взаємно доповнюючи одна одну описують зв'язок системи із середовищем.

Модель оточення системи – статична модель, що описує інші підсистеми із простору розроблюваної ПС, а модель використання системи – динамічна модель, що визначає взаємодію системи зі своїм середовищем. Ця взаємодія задається послідовністю запитів до сервісів об'єктів і одержанням відповідних реакцій системи після їхнього виконання. Дані, отримані при розробці системи і визначення взаємодій з об'єктами та оточенням, використовуються при розробленні архітектури системи з об'єктів, у тому числі зі створених у попередніх підсистемах або проектах.

Результат проектування у середовищі ООП – це ПС, у якій всі необхідні об'єкти створюються статично або динамічно за допомогою класів і відповідних операцій над об'єктами. Отримана об’єктно-орієнтована система перевіряється на показники якості за допомогою результатів тестування й збирання даних про помилки й відмови системи. Змінення методу створення об'єкта або додавання до нього нових операцій не впливає на інші об'єкти системи, які можуть бути повторно використані.

**UML-метод моделювання**

Загальна характеристика UML (Unified Modeling Language), як підход до проектування різних систем, ми розглядали на лекціях 3-6. Тут UML тільки визначимо, як мову візуального моделювання систем через подання у вигляді діаграм їхніх статичних і динамічних моделей на всіх процесах ЖЦ.

В основу методу покладено парадигму об'єктного підходу, при якій концептуальне моделювання проблеми полягає у побудові:

– онтології домену, яка визначає склад та ієрархію класів об'єктів домену, їх атрибутів і взаємозв’язків, а також операцій, які можуть виконувати об'єкти класів;

– моделі поведінки, яка задає можливі стани об'єктів, інцидентів, що ніціюють переходи з одного стану до іншого, а також повідомлення, якими обмінюються об'єкти;

– моделі процесів, що визначає дії, які виконуються при проектуванні об'єктів як компонентів.

Проектування в UML починається з побудови сукупності діаграм, які візуалізують основні елементи структури системи.

Мова моделювання UML підтримує статичні і динамічні моделі, зокрема модель послідовностей – одну з найкорисніших і наочних моделей, в кожному вузлі якої є взаємодіючі об'єкти. Всі моделі зображаються діаграмами, які ми розглядали і на лекціях 3-6, і практичних роботах 4-9, а також лабораторних роботах 1-5.

**Компонентне програмування**

З а оцінками експертів в інформаційному світі 75 % напрацювань із програмування дублюються (наприклад, програми складського обліку, нарахування зарплати, розрахунку витрат на виробництво продукції і т.п.). Більшість з цих програм типові, але кожного разу знаходяться особливості ПрО, що призводять до їх повторної розробки.

Компонентне програмування дозволяє уникнути цих проблем. Воно є подальшим розвитком ООП, заснованим на повторному використанні, специфікації компонентів і їхніх інтерфейсів, композиції та конфігурації компонентів. Зв’язки між компонентами містять у собі підтипи й еквівалентність, а об'єктні зв’язки — класи і суперкласи.

Під *компонентом* розуміють самостійний продукт, що підтримує об'єктну парадигму, реалізує окрему предметну область і може взаємодіяти з іншими компонентами через інтерфейси.

Об'єкти розглядаються на логічному рівні проектування ПС, а компоненти – це безпосередня фізична, тобто програмна реалізація об'єктів. Співвідношення між об'єктами і компонентами неоднозначне. Один компонент може бути реалізацією декількох об'єктів або навіть деякої частини системи, отриманої при проектуванні. Зворотне співвідношення, тобто компонент – об’єкт, як правило, не виконується.

Перехід до компонентів відбувався еволюційно (табл. 1.): від підпрограм, модулів і функцій. При цьому удосконалилися елементи, методи їхньої композиції і накопичення для подальшого використання.

Таблиця 1. Схема еволюції повторних елементів компонентного типу



Компоненти конструюються самостійно, як деяка абстракція, що містить у собі інформаційну частину й артефакт (специфікація, код, каркас і ін.). В інформаційній частині задаються відомості: призначення, дата виготовлення, умови застосування (ОС, середовище, платформа і т.п.).

*Артефакт* – це реалізація (implementation), інтерфейс (interface) і схема розгортання (deployment) компонента.

*Реалізація* – це код, що буде виконуватися при зверненні до операцій, визначених в інтерфейсах компонента. Компонент може мати кілька реалізацій залежно від операційного середовища, моделі даних, СКБД і ін.

Для опису компонентів, як правило, застосовуються мови ООП, зокрем Java, у якій поняття інтерфейсу і класу – базові, використовуються в моделях Javabeans і Enterprise Javabeans і в об'єктній моделі CORBA.

*Інтерфейс* відображає операції звертання до реалізації компонента, описується в мовах IDL або APL, містить у собі опис типів і операції передачі аргументів і результатів для взаємодії компонентів. Компонент, як фізична сутність, може мати множину інтерфейсів.

*Розгортання* – це виконання фізичного файлу відповідно до конфігурації (версії), з урахуванням параметрів запуску системи.

Компоненти зберігаються у вигляді класів компонентів і використовуються в компонентній моделі, композиції й у каркасі інтегрованого середовища. Керування компонентами проводиться на архітектурному або інтерфейсному рівні, між ними існує взаємний зв'язок.

Компонент описується мовою програмування, не залежить від операційного середовища (наприклад, від середовища віртуальної машини Java) і від реальної платформи (наприклад, від платформ у моделі CORBA), де він буде функціонувати

**Аспектно–орієнтоване програмування**

Аспектно-орієнтоване програмування (АОП) – це парадигма побудови гнучких до змін ПС шляхом додавання нових аспектів (функцій), що забезпечують, наприклад, безпеку, взаємодію компонентів з іншим середовищем, а компонентів із загальносистемних середовищ.

*Аспектом* може бути компонент повторного використання, фрагмент програми, що реалізує концепцію взаємодії компонентів у середовищі, захисту даних тощо. Програмна система, яка створюється з компонентів повторного використання (КПВ), об'єктів, невеликих методів та аспектів, доповнюється необхідними засобами взаємодії, синхронізації та захисту. Отже, вбудовані фрагменти наповнюють компоненти новим змістовним аспектом.

Практична реалізація аспектів, розміщених у різних частинах елементів ПС, забезпечується механізмом перетинних посилань і точками з’єднання, через які відбувається зв'язок з аспектним фрагментом для отримання визначеної додаткової функції.

В основі АОП лежить метод, подібний до методу розбиття задач ПрО на ряд функціональних компонентів, визначення необхідності використання різного роду додаткових аспектів і встановлення точок розташування аспектів в окремих компонентах, де це потрібно. Ці роботи виконуються на процесі ЖЦ процесу розробки, доповнюють реалізацію ПС засобами забезпечення взаємодії компонентів або їх синхронізації. Подібний підхід застосовується під час налагодження програм, коли додаткові фрагменти коду вбудовуються в певні точки початкової програми для видачі результатів перевірки. Якщо налагодження закінчується позитивно, ці фрагменти вилучаються. У випадку аспектів – їхні програмні фрагменти залишаються в основній програмі.

Створення кінцевої ПС в АОП виконується за технологією, що відповідає розробці компонентних систем, з тією різницею, що використані аспекти визначають особливі умови виконання компонентів у середовищі взаємодії. Аспекти можна асоціювати з виконанням різних ролей взаємодіючими особами, що наближає аспект до ролі програмного агента, який виконує додаткові функції при визначенні архітектури системи та якості компонентів.

В АОП при виробленні проектних рішень використовується механізм фільтрації вхідних повідомлень, за допомогою яких проводиться зміна параметрів і імен текстів аспектів у конкретно заданому компоненті системи. Код компонента стає «нечистим», коли його перетиняють аспекти, і при композиці з іншими компонентами загальні засоби (виклик процедур, RPC, RMI, IDL і ін.) стають недостатніми. Оскільки аспекти вимагають декларативного зчеплення описів, особливо коли їх фрагменти беруться з одних об'єктів для інших.

Один з механізмів композиції компонентів і аспектів – *фільтр композиції*, що оновлює аспекти без зміни їх функціональних можливостей. Фактично фільтрація стосується вхідних і вихідних параметрів повідомлень, які перевизначають відповідні імена об'єктів. Іншими словами, фільтри делегують внутрішнім частинам компонентів параметри, переадресовуючи раніше встановлені посилання, перевіряють і розміщують у буфері повідомлення, локалізують обмеження і готують відповідний компонент для виконання.

**Генерувальне (порождувальне) програмування**

Генерувальне програмування (generative programming) – це методи і засоби генерації сімейств систем і застосувань з окремих компонентів, аспектів, КПВ, каркасів і ін. Базис цього програмування – ООП, доповнене механізмами генерації багаторазових елементів і КПВ, а також властивостями їхньої змінюваності, взаємодії й ін. Це програмування є новим видом програмування, в ньому використовуються різні методи *інженерії складних ПрО* для розроблення сімейств ПС із різних виготовлених раніше програмних продуктів, згенерованих програмних застосувань і систем, сімейств систем, члени яких задовольняють певні показники якості.

Головний продукт інженерії ПрО – це сімейство ПС, яке генерується на основі загальної генерувальної моделі домену GDM (Generative Domain Model), що містить у собі засоби визначення окремих членів (представників) сімейства, до яких відносять предметно-орієнтовану мову DSL (Domain Specific Language), методи генерації окремих членів і їх збирання у сімейство, а також базу конфігурації для розгортання сімейства або його членів у середовищі.

Кожен член сімейства створюється з окремих компонентів. Це створення планується, контролюється й оцінюється після інтеграційного тестування на якість, а також обліку витрат на використання КПВ, у тому числі готових, узятих, наприклад, з активної бібліотеки. Елементи цієї бібліотеки – цільовий код засобів забезпечення компіляції, налагодження, візуалізації й ін. Фактично компоненти бібліотеки – це інтелектуальні агенти, що генерують нові агенти в розширюваному середовищі програмування для розв’язання конкретних задач ПрО. У ньому містяться спеціальні метапрограми, тобто програми, що генерують інші програми, ікомпоненти бібліотеки для здійснення збирання згенерованих компонентів і поповнення ними цього середовища для майбутнього створення нових членів сімейства з компонентів багаторазового використання.

Задачі простору проблем предметної області або окремих членів сімейства, як правило, визначаються різними *предметно-орієнтованими мовами*. В даному випадку термін «мова» використовується в загальному розумінні. Тобто така мова може бути подана як засіб опису специфічних понять ПрО, різних аспектів функціонування задач за допомогою операції взаємодії членів сімейств або їх складових і т.п. Поняття ПрО можуть бути поданні також процедурами, функціями, методами, як в ООП. Вони, як відомо, зберігаються в бібліотеках або вбудовуються в універсальну мову програмування (наприклад у С++, С# тощо). Коли в таку мову додаються різного типу абстракції опису різних задач ПрО, її називають *модульною предметно-орієнтованою мовою*.

Задачі можуть бути функціонального (наприклад, бухгалтерські, кадрові тощо) та системного (наприклад, захист даних, безпека, взаємодія тощо) типів. Специфікація задач домену може виконуватися декількома предметно–орієнтованими мовами, кожній з них притаманна своя специфічна мова.

До предметно-орієнтованих мов відносять такі:

– мова опису специфіки домену;

– мова опису функціональних задач домену;

– мова опису аспектів взаємодії, синхронізації компонентів у середовищі;

– мова опису захисту даних та безпеки виконання сімейства систем;

– мова опису інтерфейсів (PDL, IDL тощо).

**Предметно-орієнтована мова – DSL.** Вона вналежить до класу мов опису специфіки ПрО або домену, властивої саме цьому домену. Опис кожного члена сімейства може містить мову опису специфіки задач домену і генеруючої моделі домену GDM (Generative Domain Model), яка відображає склад домену із членів сімейства, специфікованих також у предметно-орієнтованих мовах.

Відомо, що будь-яка мова має визначену область застосування, проте мова DSL є більш спеціалізованою, ніж інші мови програмування (Паскаль, Кобол), що створювалися для розв’язання конкретних типів задач (обчислювальних, економічних). Порівняно з ними, DSL близька до описових мов, таких, як HTML, XML. Вона має специфічні особливості порівняно з мовами загального призначення, а саме:

– абстракції DSL забезпечують визначення концепцій і абстрактних понять у предметній області;

– синтаксис мови DSL може надавати засоби природного опису понять домену і запобігати синтаксичній неузгодженості, що буває при використанні мови загального призначення;

– перевірка опису в DSL вимагає статичних аналізаторів, що можуть знайти більше помилок, ніж аналізатори загального призначення, і дати повідомлення про них цією ж мовою, що є більш зрозумілим для фахівців у предметній області;

– оптимізація коду за описом в DSL базується на знаннях, що не є доступними компілятору з мови загального призначення;

– інструменти підтримки DSL потребують відповідного оточення, наприклад, середовища, редактора, контролера версій тощо.

**Сервісно-орієнтоване програмування**

Ця парадигма програмування з'явилася як наслідок розгляду програмних компонентів як готових сервісів, визначення для них інтерфейсів взаємодії в рамках нових архітектур ПЗ, зв'язаних із сервісами, у середовищі розподілених систем (CORBA, DCOM і EJB) і веб-сервісів у середовищі Інтернету. Такі архітектури отримали назву сервісно-орієнтованих архітектур – SOA (Service-Oriented Architecture) і зараз активно розвиваються разом із відповідними засобами їхньої підтримки й опису (XML, SOAP, WSDL і ін.) та механізмами взаємодії звичайних сервісів розподілених застосувань і веб-сервісів Інтернету.

Тобто у мережному середовищі поняття компонента було розвинуто до форми сервісу. *Сервіс* визначається як відкритий компонент, що може бути елементом швидкої та дешевої композиції у прикладні застосування. Сервіси пропонуються так званими провайдерами (постачальниками) – організаціями, які реалізують сервіси, надають їхні описи та іншу технічну або комерційну підтримку, якої потребують потенційні користувачі. Описи сервісів мистять у собі інформацію про їхні можливості, інтерфейси, поведінку та якісні характеристики. Завдяки такому опису користувач може знайти сервіси, вибрати потрібні йому і інтегрувати їх у свою композиційну структуру як готовий ресурс. Наведемо й інше визначення сервісу.

**Сервіс (service) –** це ресурс, що реалізує деяку функцію (у тому числі бізнес-функцію), є повторно використовуваним компонентом і містить у собі технологічно незалежний інтерфейс з іншими ресурсами. Наприклад, сервіси транзакцій, іменування, безпеки в моделі CORBA. Вони утворюють службу сервісів для створення ПС.

Архітектура SOA має форму піраміди, що складається з кількох шарів

1. Підґрунтям піраміди є базові сервіси і базові операції, а саме, публікація, виявлення, вибір і зв'язування, які націлені на створення і використання описів сервісів.

2. Шар композиції – це консолідація багатьох функціональних сервісів у єдиний складений сервіс, а саме, контроль виконання сервісів і керування потоками даних між ними; публікація подій вищого рівня шляхом фільтрації, підсумовування, кореляції подій компонентів; забезпечення цілісності сервісу та накладання обмежень на його компоненти; досягнення якості композиції сервісів, включаючи показники виконання, секретності, контролю доступу тощо.

3. Шар менеджменту сервісу, а саме, керування платформою сервісу, розгортання, ведення статистики виконання, підвищення ефективності, забезпечення прозорості ходу виконання транзакцій, відстеження стану ходу виконання тощо.

Переважна форма реалізації сервісів – це *веб-сервіси,* які зберігаються та ідентифікуються за URL-адресами і взаємодіють між собою за допомогою мережі Інтернетту шляхом віддалених викликів (Remote Procedure Call). Стрімке поширення Інтернету призвело до того, що традиційне єдине інтегроване підприємство минулих поколінь все частіше заміняється мережею бізнесів, які спільно виконують певні функції при тому, що і власність, і менеджмент розподілені між партнерами. Саме інформаційні потреби розподілених бізнесів викликали до життя веб-сервіси як адекватну форму компонентів типа КПВ.

**Веб-сервіс** має URL-адресу, інтерфейс і механізм взаємодії з іншим сервісом через протоколи Інтернету або зв'язки з іншими програмами, БД і діловими бізнес-операціями. Обмін даними між веб-сервісом і програмою здійснюється за допомогою XML-документів, оформлених у вигляді повідомлень. Веб-сервіси забезпечують розв’язання задачі *інтеграції застосувань* різної природи, будучи інструментом побудови розподілених систем. Веб-сервіс надається провайдером мережі Інтернету, який має стандартний спосіб взаємодії з розподіленими (.NET, J2EE, CORBA і ін.) і прикладними системами з отримання деякої послуги.

Основні засоби опису веб-сервісів:

– мова XML для опису і побудови SOA–архітектури;

– мова WSDL (Web Services Description Language) для опису веб-сервісів і їхніх інтерфейсів на XML, що стосується типів даних і повідомлень, а також моделей взаємодії і протоколів зв'язування сервісів між собою;

– SOAP (Simple Object Access Protocol) для визначення форматів запитів до веб-сервісів;

– UDDI (Universal Description, Discovery and Integration) для універсального опису, виявлення й інтеграції сервісів, забезпечення їхнього збереження, упорядкування ділової сервісної інформації в спеціальному реєстрі з покажчиками на конкретні інтерфейси веб-сервісів.

**Сервісно-орієнтована архітектура** – це сукупність взаємодіючих між собою сервісів і веб–сервісів і їхніх інтерфейсів. Будь-який з компонентів SOA створюється за допомогою сервісів безвідносно до конкретних технологій, за які можна брати готові застосування типу «чорна скринька». Інтеграція компонентів і сервісів в архітектуру SOA містить у собі наступні види:

– *користувальницьку інтеграцію* (user integration) для взаємодії інформаційної системи з конкретним користувачем;

– *зв'язування* застосувань (application connectivity) для забезпечення їхньої взаємодії;

– *інтеграцію процесів* (process integration) для об'єднання бізнес-процесів;

– *інформаційну інтеграцію* (information integration) для забезпечення доступу до інтегрованої інформації і даних.

При цьому до створюваної архітектури SOA висуваються наступні вимоги:

– наявність існуючих інформаційних систем і поява нових;

– поетапне впровадження нових і міграція існуючих інформаційних систем;

– стандартизація технології і реалізація інструментів для підтримки сервісних архітектур з повторним використанням застосувань і компонентів;

– використання різних моделей і систем (портали, grid-системи й ін. ).

Якщо об'єктом сервісно-орієнтованої архітектури є веб-сервіс, то застосовується дві технології, що забезпечує функціональність (Functions) і якість сервісів (Quality of service). Ці технології винесені на рівень IT-стандартів Консорціуму Всесвітнього павутиння (World Wide Web Consortium, W3C) і мають наступні рівні.

*Технологія забезпечення функціональності в*еб-сервісів має:

– транспортний рівень (transport layer) для обміну даними;

– комунікаційний рівень (service communication layer) для визначення протоколів;

– рівень опису сервісу (service description layer) і зв'язаних з ним інтерфейсів;

– рівень бізнес-процесів (business process layer) для реалізації бізнес-процесів і потоків робіт через механізми веб-сервісів;

– рівень реєстру сервісів (service registry layer), який забезпечує організацію бібліотек веб-сервісів для їхньої публікації, пошуку і виклику за їхніми WSDL-описами інтерфейсів.

*Технологія забезпечення якості* веб-сервісів має наступні рівні:

– політики (policy layer) для опису правил і умов застосування веб-сервісів;

– безпеки (security layer) для опису питань безпеки веб-сервісів і функціонування (авторизація, аутентифікація і розподіл доступу);

– транзакцій (transaction layer) для встановлення параметрів звертання до веб-сервісів і забезпечення надійності їхнього функціонування;

– керування (management layer) веб-сервісами.

**Агентне програмування**

Поняття інтелектуального і програмного агента з'явилося понад 20 років тому, їхня роль у програмній інженерії увесь час зростає. Основний теоретичний базис даного програмування – темпоральна, модальна і мультимодельна логіки, дедуктивні методи доведення правильності властивостей агентів і ін.

З погляду програмної інженерії агент — це самодостатня програма, здатна керувати своїми діями в інформаційному середовищі функціонування для одержання результатів виконання поставленої задачі і зміни поточного стану середовища. Агент має такі властивості:

– автономність – це здатність діяти без зовнішнього впливу;

– реактивність – це здатність реагувати на зміни даних, середовища і сприймати їх;

– активність – це здатність ставити мету і виконувати задані дії для досягнення цієї мети;

– здатність до взаємодії з іншими агентами (або людьми).

З інтелектуальним агентом зв'язані знання, що відображають переконання, намір, зобов'язання і т.п. Ці поняття входять у концептуальну модель і зв'язуються між собою операційними планами реалізації цілей агента. Для досягнення цілей інтелектуальні агенти взаємодіють один з одним, установлюють зв'язок між собою через повідомлення або запити і виконують задані дії або операції відповідно до наявних знань. Агенти можуть бути локальними і розподіленими. Локальні агенти виконують задані функції у певних серверах і клієнтських комп’ютерах мережі і впливають на загальний стан середовища функціонування. Розподілені агенти розміщуються в різних вузлах мережі, виконують автономно (паралельно, синхронно, асинхронно) притаманні їм функції і можуть впливати на загальний стан розподіленого середовища.

Основою агентного програмування є:

– формальна мова опису ментального стану агентів;

– мова специфікації інформаційних, часових, мотиваційних і функціональних дій агента в середовищі функціонування;

– засоби інтерпретації специфікацій агента;

– інструменти конвертування будь-яких програм у відповідні агентні програми.

Агенти взаємодіють між собою за допомогою таких механізмів, як координація, комунікація, кооперація або коаліція.

Під *координацією* агентів розуміють процес забезпечення функціонування агентів при погодженості їхньої поведінки і без взаємних конфліктів. Координація агентів визначається:

– взаємозалежністю цілей інших агентів-членів коаліції, а також від можливого впливу агентів один на одного;

– обмеженнями, що приймаються для групи агентів коаліції в рамках загального їхнього функціонування;

– компетенцією – знаннями умов середовища функціонування і ступенем їхнього використання.

Головний засіб *комунікації* агентів – транспортний протокол ТСР/IP або протокол агентів ACL (Agent Communication Languages). Керування агентами (Agent Management) виконується за допомогою сервісів: передача повідомлень між агентами, доступ агента до сервера і т.п. Комунікація агентів – це взаємодія між різними агентами через подання загальних протоколів Інтернету, а також опис повідомлень мовою HTML і декларативними або процедурними (Java, Telescript, ACL і т.п.) мовами. Кооперація агентів – це спільне виконання деяких завдань користувачів. Прикладом активної і скоординованої діяльності агентів з пошуку необхідної інформації є середовище Інтернету. У ньому агенти забезпечують доступ до інформаційних ресурсів, а також виконують аналіз інформації, інтеграцію, фільтрацію і передачу результату запиту користувачеві.

**Теоретичне програмування**

Разом з парадигмами прикладного програмування ПС продовжують розвиватися і теоретичне програмування, основано на фундаментальних дослідженнях, математичних теоріях і дисциплінах (логіка, алгебра, комбінаторика). Завдяки цьому забезпечується математичний аналіз й осмислення деяких завдань програмування, а також їх опису з використанням математичної символіки, відсутньої в прикладних методах програмування (МП). Правильність математичного аналізу треба доводити автоматизованими засобами, які розпізнають символи і забезпечують зіставлення з символікою базової мови для одержання необхідних результатів на комп'ютері.

Авторами української теоретичної школи програмування, створеної В.М. Глушковим, запропоновані нові парадигми, а саме:

– алгебраїчне та інсерційне програмування (А.А.Летичевський і ін.);

– експлікативне та номінативне програмування (В.Н.Редько, М.С.Нікітченко), які використовують логічний і математичний апарат для абстрактного конструювання програм;

– алгебро-алгоритмічне програмування (Г.О. Цейтлін), що поєднує алгебраїчний апарат і теорію алгоритмів.

**Алгебраїчне та інсерційне програмування**

Парадигма ***алгебраїчного програмування – АП*** ґрунтується на теорії переписування термів. У цій парадигмі *терми* представляють дані, а системи *переписуючих правил*, що подаються за допомогою системи рівностей, – алгоритми обчислень. Елементарний крок обчислення містить у собі включає зіставлення із зразком, перевірку умов і підстановку. Порядок вибору переписуючих правил і підтермів даного терму для зіставлення з лівими частинами рівності визначається *стратегією* переписування. По суті, стратегія визначає результат обчислень – *терм* з точністю до еквівалентності початковому терму. Власне стратегія переписування може бути описана в парадигмі більш низького рівня, наприклад, процедурній або функціональній, що зумовлює інтеграцію парадигм. На теперішній час ідея інтеграції парадигм (процедурної, функціональної, алгебраїчної і логічної) знайшла втілення в системі алгебраїчного програмування (APS), в якій використовуються спеціалізовані структури даних – *графові терми* – для представлення даних і знань про предметні області.

Основою АП є математична модель, що вміщує такі поняття:

– агент як транзитивна система, наділена поведінкою;

– поведінка агентів задається мовою АL (Action Language) за допомогою операцій, констант, граничних умов і рекурсій;

– середовище, яке складається з множини агентів і функцій занурення в нього, що позначається *env* і має параметром стан середовища й агентні вирази;

– правила розгортання функціональних виразів у прості агентні вирази;

– транзитивна система, як композиція середовища й системи взаємодіючих агентів, занурених у це середовище.

Мова дій АL, як і будь-яка інша мова, має синтаксис і семантику. Синтаксис мови задає правила опису дій, семантика – функцій, які визначаються засобами й виразами мови й ставлять у відповідність заданим виразам значення в деякій семантичній області. Різні семантичні функції можуть давати рівні абстракції й властивості програм.

**Експлікативне, номінативне програмування**

Разом з новими парадигмами програмування розробляється загальна теорія програмування, *програмологія*, яка об'єднує ідеї логіки, конструктивної математики і інформатики, *уточнює* поняття програми, самого програмування і на єдиній концептуальній основі надає загальний формальний апарат для конструювання програм.

Серед найважливіших програмних понять і принципів виділяються поняття *композиції* і *принцип композиційності*, який тлумачить програми як функції, що будуються з інших функцій за допомогою спеціальних операцій*,* названих композиціями. Принцип композиційності став основним в ***композиційному програмуванні***.

З урахуванням композиційної *експлікації* (від explication-уточнення, роз'яснення) поняття програмування було розвинуто логіко-математичною композиційної системою побудови програм, що отримала надалі назву ***експлікативного програмування*** (ЕП).Це програмування інтегрує в собі всі найбільш суттєві парадигми (стилі) програмування (структурне, функціональне, об'єктно-орієнтоване і ін.) в рамках концептуально єдиної експлікативної платформи, основу якої становлять три основні типи об'єктів: власне об'єкти, засоби побудови з одних об'єктів інших (функції) і програмологічні засоби застосування методів побудови (композиції).

Теоретичну основу ЕП становлять логіка, конструктивна математика, інформатика, композиційне програмування і класична теорії алгоритмів. Для зображення алгоритмів програм використовуються алгоритмічні мови і методи програмування: функціональне, логічне, структурне, денотаційне і ін.

Принципами експлікативного програмування є:

– розвиток поняття програми в абстрактному розумінні і поступова його конкретизація за допомогою експлікацій;

– дотримання принципу прагматичності або корисності визначення поняття програми введенням поняття *проблеми*, що ставиться для вирішення задач користувача;

– принцип адекватності орієнтований на абстрактну побудову програм і реалізацію проблеми з урахуванням інформативності даних і їх аплікативності.

Програма розглядається як функція, що виробляє вихідні дані на основі вхідних даних. Функція – це об'єкт, якому зіставляється денотат імені функції за допомогою відношення іменування (номінації); принцип дескриптивності дозволяє трактувати програму як складні дескрипції, побудовані з простих функцій і композицій відображення вхідних даних в результати на основі принципу обчислюваності. Розвиток поняття функції здійснюється за допомогою принципу композиційності, тобто складання програм (функцій) з простіших програм для створення нових програмних об'єктів зі складними дескрипціями функцій. Програми містять у собі номінативні (іменовані) дані, мовні вирази, терми і формули.

Процес розвитку програми здійснюється у вигляді ланцюжка понять: *дані –функція–ім'я функці–композиція–дескрипція*. Тріада дані–функція–композиція задає семантичний аспект програми, а дані–ім'я функції–дескрипція – синтаксичний аспект. В ЕП головними є семантичний аспект, система композицій і номінативність, що орієнтована на системний опис номінативних відношень при побудові даних, функцій, композицій і дескрипцій.

**Алгоритмічні алгебри**

Під час конструювання алгоритмів програмісти керуються в основному інтуїтивним підходом, не замислюючись над тим, чи утворюють певну систему виконувані ними операції, проте така система є, вона формально описана і називається *алгоритмічною алгеброю*. Наразі розроблено кілька алгоритмічних алгебр, найвідомішими з яких є алгебра Дейкстри, алгебра схем Янова та алгоритміка програм, досліджена у працях В.М. Глушкова і Г.О. Цейтліна.

Дослідження і побудова алгебри алгоритмів або алгоритмічної алгебри почалося з проектування логічних структур ЕОМ під керівництвом академіка В.М.Глушкова. Як результат була створена теорія *систем алгоритмічних алгебр* (САА), що потім Г.О.Цейтліним була покладена в основу узагальненої теорії структурованих схем алгоритмів і програм, названою ним *алгоритмікою*.

*Алгоритміка програм* призначена для побудови послідовних і паралельних програм зі структурних схем з використанням апарату формальних алгебраїчних перетворень і канонічних форм опису логічних і операторних виразів. Її основустановлять САА, розширені формалізмами для зображення логічних умов виконання паралельних програм, а також методами символьної мультиобробки.

Основними поняттями алгебри алгоритмів є:

– операції над множинами, булеві операції, предикати, функції й оператори;

– бінарні і *n*-арні відношення, еквівалентність, частково і цілком упорядковані множини;

– графи-схеми й операції над графовими структурами;

– операції сигнатури САА, аксіоми і правила визначення властивостей програм на основі стратегії згортання, розгортання і їх комбінацій;

– методи синтаксичного аналізу структурних програм і символьна обробка.

Об'єкти алгоритміки – моделі алгоритмів і програм, що подаються у вигляді.

*Для самостійного вивчення* *(2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Література* [1-5, 22].

*Запитання для самоперевірки.*

1. Охарактеризуйте структурний метод програмування.

2. Наведіть основні особливості і можливості об’єктно-орієнтованого програмування.

3. Які діаграми є в мові UML для візуального проектування програм?

4. Наведіть основні типи компонентів і шляхи їхнього використання.

5. Назвіть базові поняття в компонентному програмуванні.

6. Визначте основні поняття й етапи життєвого циклу у компонентному програмуванні.

7. Визначте основні елементи аспектно-орієнтованого програмування.

8. Визначте основні елементи агентного програмування.

9. Визначте об'єкти генерувального програмування і наведить призначення.

10. Наведіть теоретичні методи програмування.