КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра прикладної статистики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 124 Системний аналіз

на тему:

ПРЕДМЕТНО-ОРІЄНТОВАНІ МОДЕЛІ ПРОЦЕСІВ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Виконав студент 4-го курсу
Гончарук Андрій Ігорович

(підпис)

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Шевчук Юлія Михайлівна

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри прикладної статистики
«06» 06 2022 р.,
протокол № 11
В. О. завідувача кафедри
доц. Розора І. В.

(підпис)

Київ – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DE – Domain Engineering

DSL – Domain-Specific Language

DSML – Domain-Specific Modeling Language

ISO – International System Organization

IDL – Interface Definition Language

MDL – Method Definition Language

MDD – Metodology Domain Development

MDA – Model-Driven Architecture

RPC – Remote Procedure Call

RSL – Raise Specification Language

SE – інженерія програмного забезпечення, програмна інженерія

SQA – Software Quality Assurance (гарантія якості)

SWEBOK – Software engineering of body Knowledge

UML – Unified Modelling Language

ЖЦ – життєвий цикл

ЗКМ – загальна компонентна модель

КМ – концепторна мова

КПВ – компонент повторного використання

МП – мова програмування

ОМ – об’єктна модель

ООП – об’єктно-орієнтований підхід

ООМ – об’єктна-орієнтована методологія

ПІ – програмна інженерія

ПЗ – програмне забезпечення

ПС – програмна система

ПрО – предметна область

ВСТУП

Метою дипломної роботи є дослідження онтологічного напряму для представлення моделей предметних областей (доменів) і їх опис сучасними предметно-орієнтованими мовами, такими як DSL.

Процес побудови моделі предметної області або домену для розуміння людиною має назву концептуальне моделювання. Йому притаманна система понять домену зі своїми характерними властивостями (атрибутами), відношення та правилами поведінки. Роль концептуальної моделі полягає у тому, щоб бути посередником між професіоналами різних доменів. Ступінь її формалізації має бути достатнім, щоб забезпечувати точність та однозначність трактування понять домена.

У самому загальному випадку, онтологія – це угода про спільне використання понять і включає в себе представлення предметних знань. Звичайно, онтології представляються семантичними мережами, у яких вузлами є поняття, а дугами – зв’язки чи відносини, асоціації між ними. Тобто, онтологія, або понятійна база домену визначає склад понять, якими представляються знання у певній проблемній області (термінологію) та відносини між поняттями.

Зміст поняття може бути охарактеризованим сукупністю спільних істотних ознак тих явищ та предметів, що позначаються найменням поняття.

Одним з широко використовуваним доменом в програмній інженерії є процеси життєвого циклу, що відображені в стандарті ISO12207. Ці процеси регламентують проектування і розробку різного роду програмних систем починаючи від завдання вимог, і завершаючи його супроводженням, або експлуатацією.

В роботі будується онтологія життєвого циклу і дається її візуальне представлення засобами DSL Tools системи. Ця система слугує інструментом для візуального проектування моделей застосувань і послідовної генерації коду за описаними моделями. Побудоване візуальне представлення

розглядається як візуальний опис мовою DSL домену. Для процесів (виходячи з стандартів) виділені класи і підкласи. Представлена графічна й текстова форма опису в мові DSL, за якою виконується генерація і представлення програми в проміжній мові XML.

РОЗДІЛ І. ПОБУДОВА ОНТОЛОГІЇ ДОМЕНУ

1.1. Концептуальне моделювання проблеми

Процес побудови моделі предметної області (ПрО) або домену для моделювання її людиною має назву концептуальне моделювання. Кожну область будемо називати доменом. Йому притаманна система понять, знайому відповідним професіоналам, і яка має вважатися "загальновідомою" у рамках свого домену зі своїми характерними властивостями (атрибутами), відношеннями та правилами поведінки. Роль концептуальної моделі полягає у тому, щоб бути посередником між професіоналами та розробниками різних систем, що належать до різних доменів, наприклад, як програмісти і фахівці з економіки. Ступінь її формалізації має бути достатнім, щоб забезпечувати точність та однозначність трактування різними носіями процесів у розробці, та у той же час не надмірним, щоб бути доступним для розуміння не тільки спеціалістам за фахом, а і різними категоріями її користувачів.

Через складність та розмаїття завдань, які вирішуються за допомогою програмних систем, винайти єдину модель для всіх поки що не вдалось, однак пропонується концепція нотацій моделі, які призначені для адекватного відображення окремих аспектів проблем, а в своїй комбінації охоплюють проблеми з достатньою повнотою.

1.2. Концепція онтології домену

За основу деякого домену можна вважати його понятійну базу, тобто тему понять, за допомогою якої формулюються усі аспекти проблеми. Понятійна база визначає не тільки термінологію, якою мають користуватися носії інтересів, що приймають участь у процесі аналізу вимог, але також і суттєві відносини між поняттями та їхньою інтерпретацію.

Зміст поняття може бути охарактеризованим сукупністю спільних істотних ознак тих явищ та предметів, що позначаються найменням поняття. Сукупність явищ, охоплених поняттям, називається його обсягом. У ментальній діяльності людини застосовується ряд відношень, які дозволяють будувати похідні поняття та встановлювати між ними зв’язки. Серед таких відношень є головні:

- узагальнення як звуження істотних ознак поняття та розширення кола охоплених поняттям об’єктів, тобто їхній обсяг (наприклад, "свійська тварина" – узагальнення понять "корова", "свиня", тощо);
- конкретизація як додавання істотних ознак, завдяки чому зміст поняття розширюється, а обсяг – звужується (наприклад, "студент консерваторії" – конкретизація поняття "студент");
- агрегація як об’єднання ряду понять у нове поняття, істотні ознаки нового поняття при цьому можуть бути сумою ознак компонентів або суттєво новими (наприклад, "автомобіль" – агрегація понять "колеса", "двигун", "корпус", "кермо", тощо).

Фіксація в онтології відношень між поняттями структурує понятійний базис домену проблемної області.

Для представлення онтологій використовуються спеціальні нотації. Наприклад, діаграми "сутність-зв’язок" мають вигляд графів, у вузлах яких знаходяться поняття, а дуги відповідають відношенням між ними.

В значущих для суспільства професіях така робота лежить в руслі помітної тенденції фіксації необхідної суми професійних знань, що дозволяє проводити акредитацію відповідних навчальних програм учбових закладів та сертифікацію спеціалістів як в масштабі окремої держави, так і в міжнародному. Таким чином, розгортання робіт по створенню національних онтологій на часі є кроком до інтеграції наших фахівців у міжнародний контекст.

1.3. Методи виявлення концептів онтології.

Концепти онтології мають відображати знання, відповідні інтересам певної групи користувачів. Джерелами таких знань можуть бути термінологічні або тлумачні словники з підходящих проблемних областей, задокументовані вимоги на розробку програмних систем та інші документи. Оскільки призначенням онтології у нашому випадку є концептуальне моделювання проблеми, яку має реалізувати певний програмний продукт, то бачиться доцільним розгляд потрібних проблемних областей з позицій широко визнаного об’єктно-орієнтованого підходу до розробки програм.

Множина екземплярів зі спільним складом атрибутів та спільною поведінкою складає клас об’єктів.

Визначення класу проведено за так званим принципом приховування інформації, який можна сформулювати так: повідомляйте користувачеві тільки ті відомості, які йому потрібні для того, щоб скористатися вашими напрацюваннями, усе інше приховуйте. Такий принцип має низку переваг, серед яких основні такі:

- користувача позбавлено необхідності знати зайве, тобто непотрібне;
- те, про що йому не повідомили, він не може пошкодити, тобто здійснено захист від його неправомірних дій, як навмисних, так і випадкових;
- усе, про ще не знає користувач, можна міняти, і це не матиме на нього ніякого впливу.

Визначення об’єктів проведено згідно з наведеним принципом і складається з двох частин – видимої та невидимої. Перша з них містить усі відомості, які потрібні для того, щоб взаємодіяти з об’єктом і має назву інтерфейсу об’єкту, а друга містить подробиці його внутрішнього устрою, і вони сховані, або, як кажуть, "інкапсульовані" (тобто, знаходяться наче у капсулі). Так, наприклад, якщо нашим об’єктом є деякий прилад, що реєструє показники температури, то до видимої частини його визначення відноситься

операція показу актуального значення температури. Якщо ж до того бажано керувати гранично дозволеними діапазонами температур, як у тепличному господарстві, то ці діапазони мусять бути визначеними як видимі, тобто віднесені до інтерфейсу об’єкту. Іншим прикладом об’єкту може бути банк, зовнішня поведінка якого виглядає як можливість виконувати "видимі" операції відкриття та закриття рахунку клієнта, поповнення чи зменшення існуючого рахунку клієнта, тоді як усі інші внутрішні дії служб банку з обслуговування рахунку, що забезпечують виконання "видимих" операцій, а також усі реєстри та інші внутрішні дані банку щодо клієнтів та наявності готівки у касирів інкапсульовані, тобто клієнтові про них знати непотрібно та неможливо.

Ще одним важливим засобом визначення об’єктів є так зване успадкування (еквівалент відношення узагальнення). Кажуть, що один клас об’єктів успадковує інший, якщо він повністю містить усі атрибути та (або) поведінку. Клас, який успадковують, має назву суперкласу, а клас, що успадковує має назву підкласу. Спадковість явним способом фіксує спільні та розбіжні риси об’єктів і дозволяє явно виділити складові компоненти проблеми, які можна використати у кількох випадках шляхом побудови для них декількох класів-спадкоємців. Класи можуть утворювати ієрархії спадкоємців довільної глибини, де кожний відповідає певному рівню абстракції і є узагальненням класу-спадкоємця і конкретизацією класу, якого успадковує сам.

Для виявлення об’єктів проблемної області ми повинні, по-перше, знайти такі об’єкти, а, по-друге, надати їм унікальні та значущі найменння. Ці дії суто суб’єктивні, єдина рекомендація щодо цього полягає у переліку категорій, серед яких доцільно проводити пошук. Це наступні категорії:

- реальні предмети світу, що мають фізичне втілення, як, наприклад, Олександр Олександрович, стіл у кабінеті, Дніпро;
- абстракції фізичних предметів світу, як, наприклад, людина, тварина, літак, кабель, дім, сад;

- ролі предметів, тобто абстракції їхнього призначення або мети використання. Наприклад, для домену "університет" значущими є ролі "ректор", "декан", "викладач", "студент";
- взаємодії, тобто об’єкти, що характеризують відношення об’єктів поміж собою, як, наприклад, "контракт", "перехрестя доріг або вулиць", "угода";
- специфікації, тобто представлення правил, стандартів, критеріїв якості, обмежень користування.

Для класів об’єктів обираються значущі імена, унікальні в межах домену.

Особливим джерелом для пошуку об’єктів може бути виявлення тих робіт у межах проблемної області, які становлять окремі завдання по досягненню певних професійних цілей, які бажано комп’ютеризувати, які можуть бути реалізовані за одне звернення до системи. Таким чином відбувається послідовна декомпозиція складності кожної проблеми, яку ми можемо виявити у домені ПрО:

- складна проблема трансформується у сукупність цілей її досягнення або робіт, необхідних для її досягнення;
- кожна з цілей (або робіт) трансформується у сукупність можливих прикладів використання системи, тобто прикладів досягнення цілей (виконання робіт), що позначаються як сценарії;
- сценарії трансформуються в процесі їх аналізу у сукупність взаємодіючих об’єктів.

Визначений таким чином ланцюг трансформацій «проблема – цілі або роботи – сценарії – об’єкти» відображає ступені концептуалізації, тобто досягнення розуміння проблеми, послідовного зниження її складності та підвищення рівня формалізації моделей об’єктів.

Атрибути об’єктів. Після складання списку виявлених об’єктів, для кожного з яких потрібно визначити його характерні ознаки, або властивості, які в інформатиці називають атрибутами. Кожний атрибут є абстракцією

однієї з характеристик об’єкту, що властиві усім представникам класу об’єктів. За атрибутом закріплюємо ім’я, унікальне у межах класу.

Зв’язки об’єктів. Визначивши склад класів об’єктів домену та притаманні їм атрибути, розглянемо зв’язки (відношення), що існують між об’єктами домену.

Об’єкти одного класу можуть брати участь у бінарних, тобто попарних зв’язках з об’єктами іншого або того самого класу.

Суттєвою рисою наведених зв’язків є кількість екземплярів об’єктів, що можуть одночасно приймати в ньому участь.

Програмна система в цілому розглядається як взаємодія об’єктів, остання моделюється як обмін між об’єктами системи та зовнішніми об’єктами повідомлень про настання певних подій та даних про них.

Визначивши множину концептів, які бажано включити до онтології, на наступному кроці побудови останньої треба зв’язати концепти потрібними відношеннями, як стандартними (агрегація, асоціація, узагальнення або успадкування), так і специфічними для даної ПрО.

1.4. Життєвий цикл онтології.

Життєвий цикл онтології (для обраної предметної області) може бути представлений наступними етапами:

- первинне накопичення загальновідомих понять предметної області як тлумачного словника останньої;
- встановлення відношень між виділеними поняттями;
- накопичення розробок окремих програмних систем або інших ГОР для обраної предметної області;
- виявлення спільності накопичених розробок, узгодження їх із тлумачним словником предметної області і створення концептуальної моделі домена;

- представлення вимог для нової розробки під керуванням онтології домена;
- модифікація онтології і концептуальної моделі домена відсутніми для нової розробки можливостями.

1.5. Метамоделі опису онтологій.

Мови визначення онтологій пройшли певний період розвитку, починаючи з кінця дев'яностих років, коли роль і вплив онтологій були досить обмеженими, разюче змінюється розумінням, що концептуальна здійсненна модель прикладного домену забезпечує істотне додаткове значення для усіх видів прикладних сценаріїв. Очевидно, головний поштовх онтологіям, дало передбачення SemanticWeb [BLHL 01]. У SemanticWeb онтології забезпечують концептуальне підкріплення для створення семантики метаданих. Завершенням розвитку можна вважати широко відому мову з назвою Web Ontology Language (OWL) для публічного доступу та обміну онтологіями в Інтернеті. OWL належить до спектру семантичних мов розмітки, розвиток яких проводиться міжнародним консорціумом W3C і структурно побудована як розширення словника RDF.

Онтологія OWL являє собою послідовність аксіом і фактів, які можуть супроводжуватися рекомендаціями щодо потенціального включення інших онтологій. Онтології OWL є документами WEB і можуть імпортуватися через URI. Онтології також мають нефункціональний компонент, який може використовуватися для записів авторства, іншої нефункціональної інформації, пов'язаної з онтологією.

Онтології включають інформацію про класи, властивості і ресурси, кожний з яких може мати ідентифікатор (ID), що є посилання URI. Онтології можуть також посилатись на XML-схему datatypes, за допомогою назви для datatype:

<Datatype ID>:: = <URI посилання>.

Є два види фактів OWL. Перший вид факту представляє інформацію про специфічний ресурс у формі класів, що складають ресурс, та властивостей і значення цього ресурсу. Ресурсу можна надати індивідуальний ID, це позначить, що до цього ресурсу можна звернутися, але допускаються також анонімні ресурси (пробіл (blank) у термінах RDF), на які безпосередньо не можуть бути посилання.

Аксіоми використовуються, щоб зіставити клас і властивість ID з частковими чи повними специфікаціями їхніх характеристик, і надати іншу логічну інформацію про класи і властивості. Кожна аксіома класу містить сукупність більш загальних класів; і сукупність локальних обмежень властивості у формі конструкцій обмеження. Конструкція обмеження може визначити локальний діапазон властивості, а також скільки значень дозволяється та сукупність необхідних значень. Клас може бути еквівалентом, підмножиною або перетинанням більш загальних класів з обмеженнями. У OWL аксіома класу містить сукупність описів, які можуть мати вигляд узагальнених класів, обмежень, наборів ресурсів, булевих комбінацій описів. Класи можуть також бути визначені перерахуванням, бути подібними.

Властивості можуть співпадати, бути підвластивостями інших властивостей; можуть бути зроблені функціональними, зворотними функціональними чи перехідними; для них можна задавати глобальні домені значень та діапазони. Однак частіше інформацію про властивості виражають в обмеженнях, що представляють локальних діапазон і інформацію про кількість визначених елементів.

РОЗДІЛ ІІ. ОНТОЛОГІЯ МОДЕЛЕЙ ЖИТТЄВОГО ЦИКЛУ ДЛЯ РОЗРОБКИ ПРОГРАМНИХ СИСТЕМ

Модель життєвого циклу – це схема виконання робіт та завдань на процесах, що забезпечують розробку, експлуатацію та супровід програмного продукту, та відображає життя ПС, починаючи від формулювання вимог до неї до припинення її використання.

Історично в цю схему робіт включають:

- розробку вимог або технічного завдання,
- розробку системи або технічного проекту,
- програмування або робоче проектування,
- пробну експлуатацію,
- супровід та поліпшення,
- зняття з експлуатації.

Вибір і побудова моделі ЖЦ ПС базується на концептуальній ідеї проектованої системи, її складності і стандартів, що дозволяють формувати схему виконання робіт на розсуд розробника та замовника. Модель ЖЦ розбивається на процеси реалізації, які повинні включати окремі роботи та завдання, реалізовані в даному процесі, і при їхньому завершенні здійснювати перехід до наступного процесу моделі.

При виборі загальної схеми моделі ЖЦ для конкретної предметної області, вирішуються питання включення або невключення окремих робіт, дуже важливих для створюваного виду продукту. На сьогодні основою формування нової моделі ЖЦ для конкретної прикладної системи є стандарт ISO / IEC 12207, який включає повний набір процесів (понад 40), що охоплює всі можливі види робіт і завдань, пов’язаних з побудовою ПС. [8]

З цього стандарту можна вибрати тільки ті процеси, які найбільше підходять для реалізації даного ПС. Обов’язковими є основні процеси, які присутні у всіх відомих моделях ЖЦ. У залежності від цілей і завдань предметної області вони можуть бути поповнені процесами з додаткових або

організаційних процесів або підпроцесів цього стандарту. Наприклад, рішення питання включення в нову модель ЖЦ процесу забезпечення якості компонентів і системи в цілому або визначення набору перевірочних (верифікаційних) процедур для забезпечення правильності та відповідності розроблюваного ПС (валідація) заданим вимогам, а також процесу забезпечення можливості внесення змін до вимог або в компоненти систем і т. д. [9]

Процеси, включені в модель ЖЦ, призначені для реалізації унікальної функції ЖЦ і можуть залучати інші процеси для виконання спеціалізованих можливостей системи (наприклад, захист даних). Інтерфейси між двома будь-якими процесами ЖЦ повинні бути мінімальними і кожен з них прив'язаний до архітектури системи. Якщо робота чи завдання потрібно більш ніж одному процесу, то вони можуть стати процесом, використовуваним одноразово або протягом життя системи. Кожен процес повинен мати внутрішню структуру, встановлену відповідно до того, що повинно виконуватися на цьому процесі.

Процеси моделі ЖЦ орієнтовані на розробника системи. Він може виконувати один або декілька процесів і процес може бути виконаний одним або кількома розробниками, при цьому один з них є відповідальним за один процес або за все на моделі, навіть якщо окремі роботи виконує інший розробник.

Створювана модель ЖЦ ув'язується з конкретними методиками розробки систем і відповідними стандартами в галузі програмної інженерії. Іншими словами кожен процес ЖЦ підкріплюється вибраними для реалізації завдань коштів і методів. [1]

Важливу роль при формуванні моделі ЖЦ мають організаційні аспекти:

- планування послідовності робіт та строків її виконання,
- підбір та підготовка ресурсів (людських, програмних і технічних) для виконання робіт,
- оцінка можливостей реалізації проекту в задані терміни і вартість.

Впровадження моделі ЖЦ в практичну діяльність зі створенням програмного продукту дозволяє впорядковувати відносини між суб'єктами процесу розробки ПС і враховувати динаміку модифікації вимог до проектів і систем.

Ці та інші питання послужили джерелом формування різних видів моделей ЖЦ, заснований на процесному підході до розробки програмних проектів. Основними серед них зарекомендували себе в практиці програмування наступні: каскадна, галактика, інкрементна, еволюційна та стандартизована.

2.1. Каскадна модель ЖЦ.

Однією з перших почала застосовуватися каскадна або водоспадна модель, в якій кожна робота виконується один раз і в тому порядку, як вони представлені в схемі моделі ЖЦ. Тобто робиться припущення, що кожна робота буде виконана настільки ретельно, що після її завершення і переходу до наступного етапу повернення до попереднього не буде потрібно. Розробник перевіряє проміжний результат різними відомими методами верифікації і фіксує його в якості готового еталону для наступного процесу.

На рис.2.1. показана каскадна модель. У ній повернення до початкового

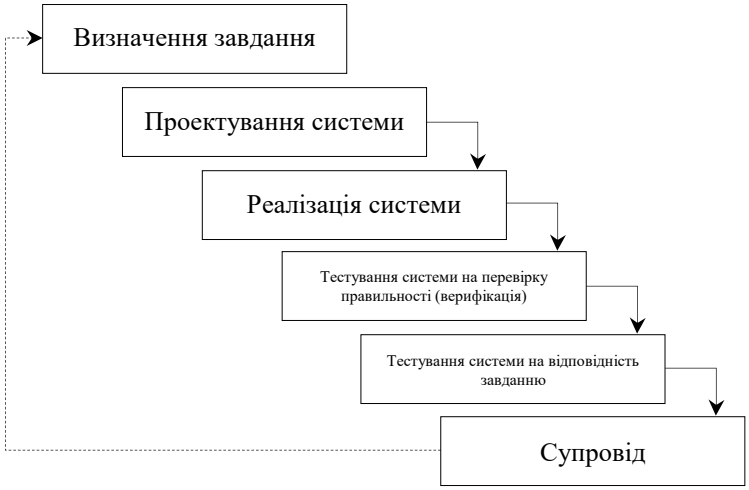

Рис. 2.1 Каскадна модель ЖЦ розробки програмних систем

процесу робіт передбачається після супроводу при поверненні на початковий процес. [1]

Відповідно до даної моделі роботи і завдання процесу розробки зазвичай виконується послідовно, як це представлено на схемі. Однак допоміжні та організаційні процеси (контроль вимог, показників якості та ін.) зазвичай виконуються паралельно з процесом розробки.

Цінність такої моделі полягає у фіксації послідовних процесів розробки програмного продукту. Недоліком цієї моделі є те, що в основу її концепції покладено модель фабрики, де продукт проходить стадії від задуму до виробництва, потім передається замовнику як готовий виріб, зміна якого не передбачена, хоча можлива заміна на інший подібний виріб у разі рекламації або деяких її деталей, що вийшли з ладу.

При такому підході необхідно враховувати наступні фактори ризику:

- вимоги недостатньо добре представлені;
- система занадто велика за розміром, щоб бути реалізованою в цілому;
- швидкі зміни в технології та у вимогах;
- обмежені ресурси (людські, програмні та ін.);
- отриманий продукт може виявитися непридатним для використання через неправильне розуміння вимог або функцій системи, а також через недостатнє тестування.

Переваги реалізації системи за допомогою каскадної моделі наступні:

- всі можливі системи реалізується одночасно;
- застосовується у разі, якщо стара система повинна бути повністю замінена.

Каскадну модель можна розглядати як модель ЖЦ, придатну для створення першої версії ПЗ для перевірки реалізованих в ній функцій. При супроводженні та експлуатації можуть бути виявлені різного роду помилки, які будуть виправлятися розробником, починаючи з першого процесу даної моделі.

2.2. Спіральна модель ЖЦ.

Виходячи з можливості внесення змін як до процесу, так і в створюваний проміжний продукт була створена спіральна модель.

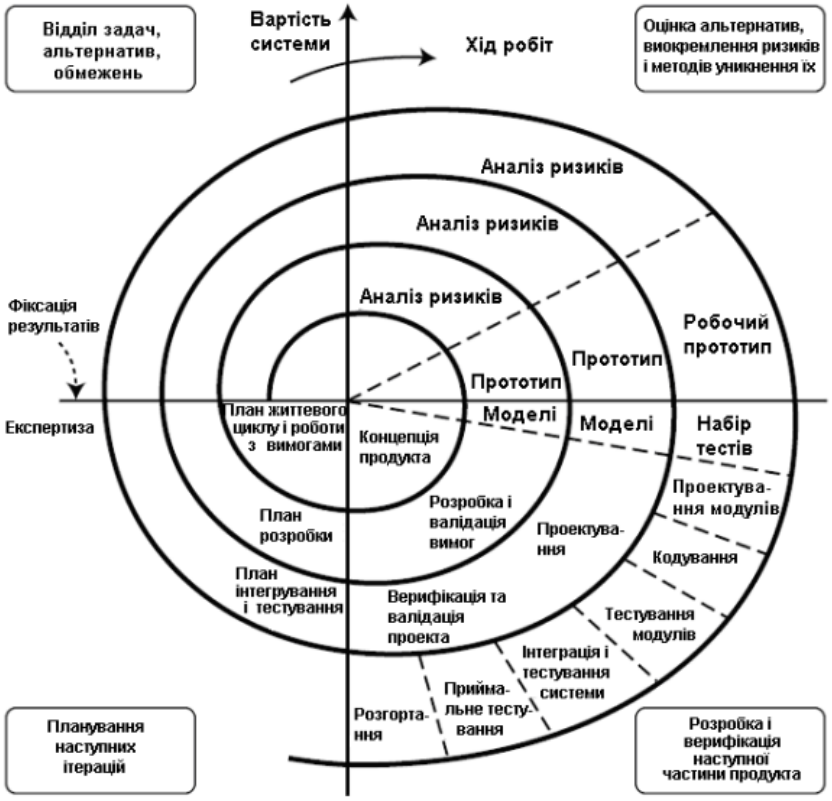

Рис. 2.2. Спіральна модель ЖЦ розробки програмний систем

Це припущення орієнтоване на задоволення потреби змін відразу, як тільки буде встановлено, що створені артефакти або елементи документації (опис вимог, проекту, коментарі різного виду та ін.) не відповідають дійсному стану розробки після внесення деяких змін. Дана модель ЖЦ допускає аналіз продукту на витку розробки, його перевірку, оцінку його правильності і прийняття рішення рухатися на наступний виток або опуститися на нижній для доопрацювання.

Відмінність цієї моделі від каскадної полягає в можливості спіральної моделі забезпечувати багаторазове повернення до процесу формулювання вимог і до повторної розробки з будь-якого процесу виконання робіт.

На зображеній спіральній моделі (рис. 2.2), кожен виток спіралі відповідає одній з версій розробки системи. При необхідності внесення змін до системи на кожному процесі витка, обов’язково вносяться зміни до попередньо зафіксованої вимоги, після чого відбувається повернення на

попередній виток спіралі для продовження реалізації нової версії системи з урахуванням змін.

2.3. Стандартизована модель системи.

Типовий ЖЦ розробки починається з формулювання ідеї або потреби, проходить всі процеси розробки, виробництва, експлуатації та супроводу системи. ЖЦ в практиці програмування зазвичай ділиться на етапи, процеси. Кожен процес характеризується видами діяльності та завданнями, які виконуються на ньому. Перехід від одного процесу до іншого повинен бути санкціонований (визначені вхідні та вихідні дані).

Модель загального стандартизованого ЖЦ, як правило, включає в себе наступні процеси (табл.2.1.):

- визначення вимог;
- розробка (проектування);
- верифікація, валідація, тестування;
- виготовлення;
- експлуатація;
- супровід.

Таблиця 2.1. Процеси життєвого циклу в стандарті ISO/IEC 12207

№ п/п	Процес (підпроцес)
1. Категорія «Основні процеси»	
1.1.	Замовлення (договір)
1.1.1.	Підготовка замовлення, вибір постачальника
1.1.2.	Моніторинг діяльності постачальника, приймання споживачем
1.2.	Постачання (придбання)
1.3.	Розробка
1.3.1.	Виявлення вимог
1.3.2.	Аналіз вимог до системи
1.3.3.	Проектування архітектури системи
1.3.4.	Аналіз вимог ПЗ до системи
1.3.5.	Проектування ПЗ
1.3.6.	Конструювання (кодування) ПЗ

1.3.7.	Інтеграція ПЗ
1.3.8.	Тестування ПЗ
1.3.9.	Системна інтеграція
1.3.10.	Системне тестування
1.3.11.	Інсталяція ПЗ
1.4.	Експлуатація
1.4.1.	Функціональне використання
1.4.2.	Підтримка споживача
1.5.	Супроводження
2. Категорія «Процеси підтримки»	
2.1.	Документування
2.2.	Керування конфігурацією
2.3.	Забезпечення гарантії якості
2.4.	Верифікація
2.5.	Валідація
2.6.	Загальний огляд
2.7.	Аудит
2.8.	Вирішення проблем
2.9.	Забезпечення застосовності продукту
2.10.	Оцінювання продукту
3. Категорія «Організаційні процеси»	
3.1.	Керування
3.1.1.	Керування на рівні організації
3.1.2.	Керування проектом
3.1.3.	Керування якістю
3.1.4.	Керування ризиком
3.1.5.	Організаційне забезпечення
3.1.6.	Вимірювання
3.1.7.	Керування знаннями
3.2.	Удосконалення
3.2.1.	Упровадження процесів
3.2.2.	Оцінювання процесів
3.2.3.	Удосконалення процесів

Даній моделі відповідають всі види діяльності, які починаються з розробки ідеї проблеми або концепції програмного продукту і закінчуючи його виготовленням. Стандарт ISO/IEC 12207 об’єднує ці види діяльності в основні, організаційні та допоміжні процеси, які і складають ЖЦ ПЗ. [1]

Процеси ЖЦ придбання, постачання і розробки використовуються для аналізу і визначення системних вимог, рішень верхнього рівня проектування

системи, включаючи ПЗ. Процес розробки може бути використаний для аналізу, демонстрації, валідації, тестування, прототипування вимог і проектних рішень.

На цьому етапі проектування розробляється технічне, програмне, організаційне забезпечення системи, а також проектуються, розробляються, інтегруються, тестуються і оцінюються компоненти системи. Результатом цього процесу є система, яка відповідає потрібному продукту.

Вихідний стандарт розроблений так, щоб його можна було застосувати повністю або частково. Процеси, дії і завдання основних процесів відбираються, адаптуються і застосовуються для розробки або модифікації ПЗ. Процес проектування може включати одну або декілька ітерацій процесу розробки. Результатом є основні вимоги до ПЗ, проект та його реалізація.

Якщо ПЗ, що розробляється є частиною системи, то можуть знадобитися всі дії процесу розробки, а якщо ПЗ – автономне, то всі дії на рівні системи можуть не знадобитись.

Метою процесу є виробництво та установка працюючої системи у замовника для супроводу. Для ПО даний процес полягає в копіюванні виготовленого ПЗ та документації на відповідні носії для різних користувачів. До видів діяльності на процесі належить – управління реалізацією і конфігурацією. Інші допоміжні процеси та дії можуть застосовуватись при необхідності.

Виготовлена система передається замовнику або продається покупцям. Період розгортання системи починається з поставки першої її версії замовнику. Продаж системи починається з першої версії системи, яка поставляється всім покупцям і закінчується вилученням з ринку. Інші процеси (придбання, постачання і розробки) можуть використовуватись при інсталяції і перевірці розробленої або модифікованої системи.

Процес експлуатації включає використання системи користувачами / покупцями і закінчується, коли система більше не задовольняє користувачів і вона видаляється з експлуатації.

Під час супроводу система модифікується, внаслідок виявлених помилок і недоліків у її розробці або за вимогами користувача, який бажає її адаптувати до нового середовища або вдосконалити окремі функції системи. Даний процес включає забезпечення логічної, технічної та програмної документації користувачеві.

Процес видалення системи означає зняття її з обслуговування, видалення її архівів і носіїв кодів системи. [1]

2.4. Характеристика процесів стандарту.

Процеси даного стандарту розбиті по групах: основні, допоміжні та організаційні.

До основних процесів стандарту відносяться:

- придбання (acquisition);
- поставки (supply);
- розробки (development);
- експлуатації (operation);
- супроводу (maintenance).

Процес придбання ініціює ЖЦ ПЗ і визначає дії організації – організації-покупця (або замовника), яка отримує автоматизовану систему, програмний продукт або сервіс. Процес постачання визначає дії підприємства – постачальника, що постачає покупця системою, програмним продуктом або сервісом. Процес розробки визначає дії підприємства – розробника, що розробляє програмний продукт. Процес експлуатації визначає дії підприємства – оператора, яке забезпечує обслуговування системи (ПЗ) в процесі її експлуатації користувачем (консультація користувача, вивчення його потреб з точки зору задоволення його системою, тощо). Процес супроводження визначає дії організації, що виконує супровід програмного продукту (управління модифікаціями, підтримка поточного стану та

функціональної придатності, інсталяція та видалення програмного продукту на обчислювальній системі користувача).

Допоміжні процеси підтримують реалізацію основних процесів і забезпечують необхідну якість ПЗ. Вони ініціюються іншими процесами. До допоміжних процесів стандарту відносяться:

- документування (documentation);
- управління конфігурацією (configuration management);
- забезпечення якості (quality assurance);
- верифікації (verification);
- спільного аналізу (оцінки) (joint review);
- аудиту (audit), процесу вирішення проблем (problem resolution).

До організаційних процесів стандарту належать:

- управління (management);
- створення інфраструктури (infrastructure);
- удосконалення (improvement);
- навчання (training).

За кожний процес стандарту відповідає певний учасник розробки або керівник. Для кожного з процесів стандарту визначено види діяльності - дії та завдання, які в нього входять, визначено сукупність результатів видів діяльності та завдань, а також деякі специфічні вимоги. У табл. 2.1. наведено загальна кількість визначених у стандарті процесів (17), дій (74) та завдань (232).

РОЗДІЛ ІІІ. МОВА DSL ДЛЯ ОПИСУ ДОМЕНІВ ЖЦ

3.1. Загальна характеристика мови.

Спеціалізована мова предметної області (DomainSpecificLanguage) – це мова, розроблена для того, щоб бути корисною для вирішення вузького кола специфічних завдань. За допомогою інструментів DSL можливо створення спеціалізованих інструментів моделювання шляхом визначення нової мови моделювання і реалізації.

Інструменти DSL можуть бути використані для побудови власних візуальних редакторів, пристосованих для конкретної предметної області. Для цього необхідно створення метамоделі мови створюваного редактора за допомогою DSL. Використовуючи отриману метамодель, в роботі створюється візуальний редактор. Також з використанням створенної метамоделі можлива генерація звітів по створюваному новому редактору моделі. Для моделей, що описують конструкції мов програмування, можлива генерація початкового коду.

Мова DSL має виразну особливість, спрямовану на відображення специфіки предметної області. Тоді як мови загального призначення (Java, C++ і інші) створювалися для задоволення будь-якого типу програм. Мова DSL не є новим винаходом, оскільки в неї були вбудовані загальні абстракції програмування. І хоча мова DSL створюється багато років для кожних нових ПрО, систематичне її застосування для специфікації особливостей ПрО ще не є загальноприйнятою практикою.

Будь-яка мова має свою визначену область застосування, мова DSL є більш спеціалізованою, чим інші мови програмування (Фортран або Кобол), які створювалися з визначеною метою. Порівняно з ними DSL близька до спеціалізованих мов, таких, як HTML, XML.[2]

3.2. Характеристика мови DSL опису специфіки ПрО.

Порівняно з мовами загального призначення ця мова має:

- абстракції DSL, які забезпечують визначення концепцій і абстрактних понять із предметної області;
- синтаксис, призначений для природного представлення понять домену і запобігання синтаксичній неузгодженності понять ПрО;
- опис в DSL вимагає наявності спеціалізованих статичних аналізаторів для перевірки синтаксису, щоб знайти помилки в описі специфікації моделі, і подати про них відомості мовою DSL;
- оптимізація коду за описом в DSL базується на знаннях про цю область і це не доступно компіляторам мов загального призначення;
- інструменти підтримки DSL вимагають оточення, наприклад, середовища, редактора, засобу контролю версій і т.ін;
- предметні абстракції в мовах програмування базуються на використанні бібліотек програм і КПВ, визначених користувачем функцій, класів і структур даних.

Специфікована в DSL модель ПрО є загальною моделлю генерації домену GDM, що відбиває:

- поняття, характеристики ПрО і членів сімейства в просторі проблем;
- набір членів сімейства і їхніх специфікацій у мовах типу DSL, RSL, CSP, і ін.;
- задачі ПрО в просторі задач для їхньої реалізації компонентами і з наступним збиранням їх у конфігурацію визначених членів сімейства;
- знання про конфігурацію (Configuration knowledge) відбивають характеристики членів сімейства і їхнє об’єднання в конфігурації;
- модель характеристик (feature models) відображає загальні, змінювані і незмінні характеристики членів сімейства, а також правила

конструювання систем сімейства з урахуванням їхньої взаємозалежності і залежності від КПВ;

- архітектуру (каркас) сімейства систем;
- реалізацію компонентів архітектури у мовах програмування.

При проектуванні каркаса або моделі ПрО у вигляді моделі GDM можуть задаватися механізми змінювання, синхронізації, безпеки з використанням аспектного програмування.[4]

Головною частиною моделі ПрО є моделі характеристик, що призначені для подання вимог до сімейств ПС, різних характеристик систем, що входять до складу ПрО та застосування КПВ. Ці моделі поділені на дві категорії:

- моделі характеристик FODA (Feature-oriented Development Architecture), FORM (Feature-oriented Reuse Method), RSEB (Reuse-Driven Software Engineering Business), FeatureRSEB (з інтеграцією у RSEB-діаграм характеристик з FODAcom) тощо,
- моделі варіантів сценаріїв використання (Use Case Variants).

У першу категорію потрапляють традиційні методології моделювання, використовувані при побудові ПС, у другу – методології, засновані на аналізі сценаріїв діяльності потенційних користувачів ПС у ПрО, що будуються за допомогою (Sequence diagrams та Activity diagrams).

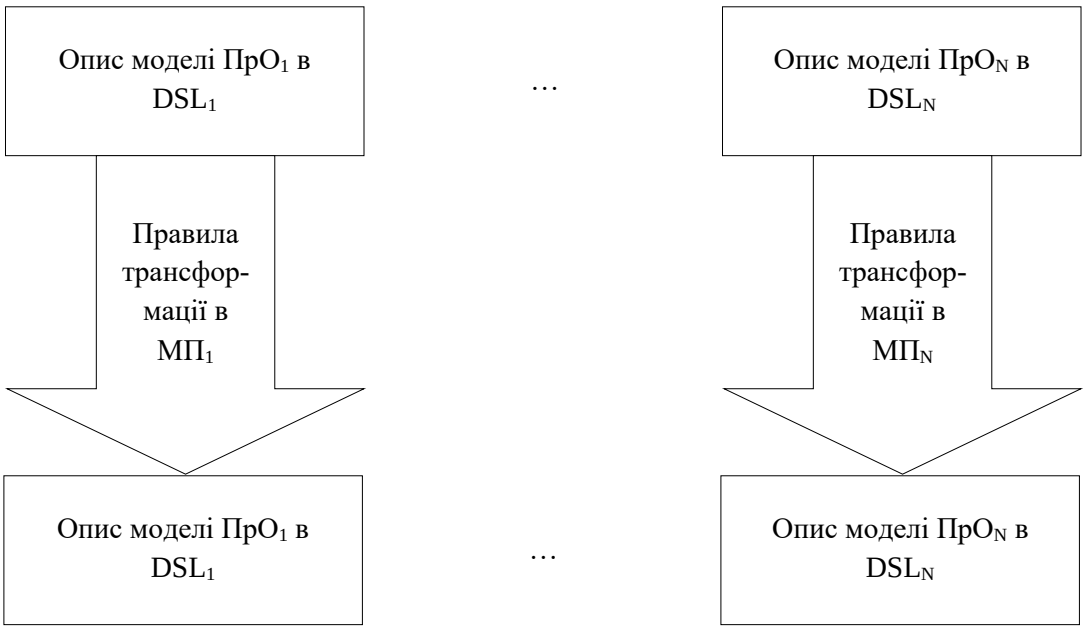

Рис.3.1. Схема трансформації моделей прикладних систем

Про трансформацію опису моделей ПрО в DSL. Як було сказано раніше, модель предметної області МПрО є загальною моделлю GDM домену, який складається з декількох моделей окремих прикладних систем $МПрО_1$, $МПрО_2$,…, $МПрО_N$. Кожна з цих моделей відображає поняття і специфіку відповідної системи із сімейства систем домену. На їхньому перетині існують загальні поняття, характеристики і обмеження, які відображаються у моделі характеристик ПрО. Опис кожної моделі $МПрО_i$ виконується відповідною проблемно-орієнтованою DSL_i-мовою. Цей опис трансформується безпосередньо у відповідному $МП_i$, тобто мову реалізації, або у іншу DSL_j-мову, яка потім трансформується у МП. [3]

Після трансформації опис моделей $МПрО_1$, $МПрО_2$,…, $МПрО_N$, що був отриманий у МП, транслюється до вихідного коду того середовища, де цей код буде функціонувати. Крім того, вибрані КПВ і знову розроблені компоненти для окремих функцій простору проблем також описуються відповідними МП, і отже, можуть розглядатися як елементи реалізації завдань у просторі задач. Перебудова до вихідного коду залежить також від платформи, концепція якої буде розглядана нижче.

Даний підхід подання моделей ПрО відповідає відомій модельно-керованій розробці MDD (Model Driven Development). Відповідно цей моделі архітектури системи моделюються на двох рівнях – платформа рівня незалежного за моделлю PIM (Platform Independent Model) і платформа залежного рівня за моделлю PSM (Platform Specific Models). Концепції дворівневого моделювання архітектури MDA (Model Driven Architecture) і відображення PIM → PSM безпосередньо відповідають наведеній ідеології побудови сімейства ПрО, які базуються на відображенні опису моделей прикладних систем (Application model) у DSL-мовах у МП.

Відповідно до підходу MDD моделі прикладних систем (application model) у моделі сімейства ПС відображають члени сімейства. Вони мають спільні функції, але розрізняються платформами реалізації. Вибору альтернативної платформи відповідає точка варіантності у сімействі систем.

Ця точка невидима на рівна конкретної ПС і при керуванні варіантами платформ з неї починається автоматично виконання шляхом трансформації PIM→PSM, розробник не бере участі у цьому. [4]

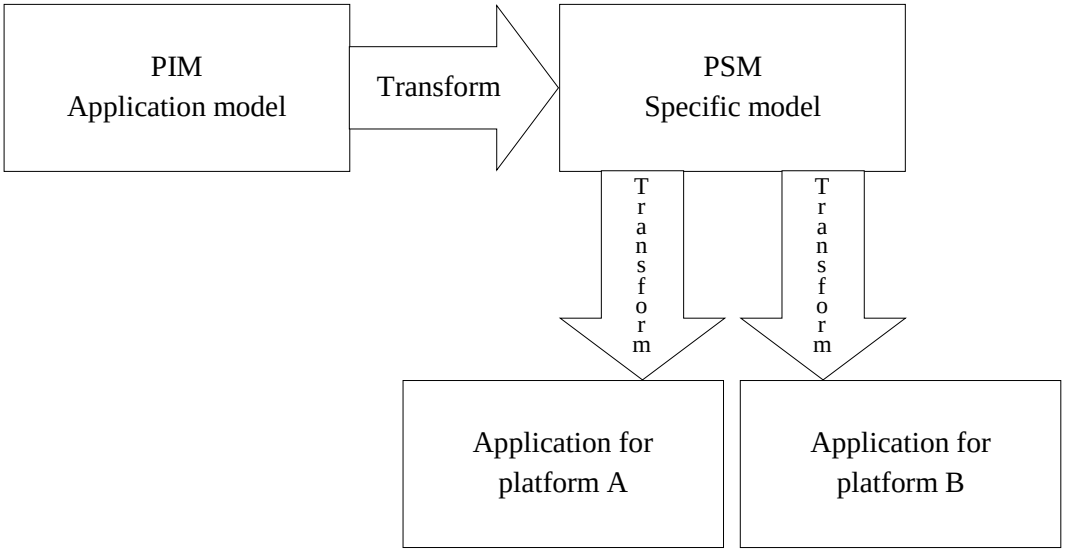

Рис. 3.2. Схема трансформації прикладної моделі для платформи

Зазначимо, що члени сімейства розрізняються не лише на рівні платформи реалізації, а й на рівні функцій ПС, вимог до якості, які реалізують альтернативні концепції. У такому випадку до моделі ПрО додається точка, де можуть бути подані необхідні альтернативні концепції.

Моделювання особливостей ПрО – спосіб відображення загальних і змінних специфічних властивостей кожної прикладної проблеми у відповідних моделях, які входять у модель ПрО для сімейства. До особливостей моделей належать:

- конкретні особливості, такі як зберігання інформації, або сортування, які можуть бути реалізовані в індивідуальних спеціальних компонентах;
- аспектні особливості, такі як збирання даних, синхронізація, які можуть впливати на кількість компонентів і можуть бути перетворені в модулях, які використовують аспектні технології;
- абстрактні особливості, такі як вимоги до швидкодії, які зазвичай відображають певні конфігурації компонентів або аспектів;

- групові особливості можуть відображати як рівні погляди так і загальні сумісні інтерфейси, або можуть мати організаційні наміри без будь-яких вимог.

Моделювання особливосте ПрО дає початок новому підходу в межах генеруючого програмування. На початковому етапі розробки сімейства ПС особливості можуть забезпечувати основу для визначення меж сімейства систем, що містять у собі запис і оцінки інформації, яка важлива, наприклад, для виходу продукту на новий ринок, або для того, щоб залишитись на існуючому, а також особливості, пов’язані з технологічними ризиками, щодо ціни розробки проекту в майбутньому, тощо. Модель особливостей створюється при аналізі проблемної області і є початком при розробці архітектури сімейства систем і опису їх мовою DSL. [5]

Розробка архітектури починається з простору рішень і є їхньою проекцією на моделі особливостей. Цей простір конкретизується на конкретних і аспектних особливостях і має бути виконаним у вигляді компонентів і аспектів. Вільні архітектурні шаблони можуть бути прикладними, які виражені в моделі особливостей, і вони повинні бути реалізовані в архітектурі системи.

Методологія розробки домену. На основі мови DSL створено методологію розробки домену MDD, що базується на моделі GDM (Generative Domain Model), кожний член сімейства для якого специфікується предметно-орієнтованими мовами. Методологія містить у собі:

- модельно-керовану архітектуру MDA, що використовує мову моделювання UML для подання архітектури системи і її конкретні профілі для різних платформ;
- модельно-інтегровану обробку MIC (Model-Integrated Computing) для реалізації елементів системи і їхніх зв’язків за допомогою засобів предметно-орієнтованої мови моделювання DSML (Domain-Specific Modeling Language) і перетворення цього опису в платформозалежні артефакти.

Архітектура MIC поєднує:

- предметно-орієнтовані мови моделювання DSML, за допомогою яких формалізується структура, поведінка і вимоги до застосувань усередині ПрО, а також визначаються зв'язки між її поняттями, семантика й обмеження щодо використовування поняття;
- трансформаційні процесори і генератори, що аналізують задані аспекти моделі і синтезують вихідний код у XML-опис, включаючи схеми розгортки, узгодження між реалізаціями і зафіксованими у моделі вимогами до функцій системи з відповідною якістю.

При проектуванні каркаса або моделі ПрО у вигляді моделі GDM можуть задаватися механізми змінювання, синхронізації, безпеки з використанням аспектного програмування.

Технологія розробки сімейства програм вміщує три види базових процесів:

- розробка ПрО і унікальних, одиночних програм;
- інженерія повторного використання ресурсів;
- менеджмент домену.

Розробка ПрО належить до конвеєрної з повторним використанням компонентів. Для ПрО планується створення базових ресурсів, що можуть повторно використовуватися: компоненти, генератори, DSL-описи, моделі аналізу, КПВ, документація й ін. Розробка предметної області – це більш складний виробничий процес, що містить у собі такі загальні етапи: аналіз, проектування і впровадження в ПрО одиночних програм або КПВ.

Аналіз області містить у собі аналіз усього сімейства, яке треба побудувати, визначаючи в ньому загальні і відмінні риси і створюючи структурні і поведінкові специфікації сімейства. Аналіз ПрО починається з вибору вимог і їхньої специфікації для системи і членів сімейства. Специфікація вимог – це вхідні дані для ручного або автоматичного створення домену з готових ресурсів.

Інженерія повторного використання – це пошук і аналіз КПВ для їхнього повторного використання під час розв’язання задач ПрО. Вони відображаються в загальній архітектурі сімейства і в архітектурі всіх членів (тобто прикладних систем) сімейства цієї ПрО. Такими членами можуть бути одиночні прикладні системи, що вироблені методами прикладної інженерії, або готові КПВ.

Впровадження або реалізація ресурсів для повторного використання в ПрО містить у собі готові компоненти, одиночні програми, генератори, DSL-описи та ін. Об’єднання цих ресурсів у зв’язану сукупність виконується за допомогою генераторів або конфігураторів готових ресурсів. Згенеровані продукти сімейства можуть мати не програмні артефакти, наприклад, інструкції з користування DSL, інструкції з використання домену й ін.

Менеджмент домену – це керування конвеєрною розробкою з повторним використанням ресурсів, що містить у собі планування і контроль підбору типових для ПрО ресурсів, їхню оцінку і перевірку відповідності вимогам. У задачу менеджменту входить також перевірка можливості застосування тих чи інших готових ресурсів для реалізації специфіки ПрО і програмування компонентів простору задач відповідно до потреб клієнтів домену.

Таким чином, інженерія ПрО складається з таких процесів:

- аналіз ПрО, виявлення об’єктів і зв’язків між ними;
- визначення області дій об’єктів ПрО;
- визначення загальних функціональних і змінюваних характеристик, побудова моделі характеристик, що встановлює залежність між різними членами сімейства;
- створення базису для інженерії виробництва конкретних прикладних членів сімейства з механізмами змінювання незалежно від засобів їхньої реалізації;
- підбір і підготовка компонентів багаторазового застосування для задач ПрО;

- генерація окремих членів сімейства або домену в цілому.

Процеси в цій схемі забезпечують формування моделі ПрО і моделі характеристики, як елементів простору проблем. Вони трансформуються в архітектуру системи й опис її компонентів. При цьому проводиться підбір готових компонентів і їхня генерація для одержання програм, що розв'язують задачу ПрО у просторі задач. Таким чином, дана схема приводить до генерації доменної моделі GDM для сімейства ПС і породженню готових підсистем або окремих членів сімейства.

Для забезпечення гнучкості і змінюваності деяких членів сімейства технологічно схема інженерії ПрО може дати додаткові етапи:

- коректування процесів виробництва в зв'язку з включенням у систему нових проектних рішень або при зміні складу КПВ;
- моделювання змінності і залежностей між компонентами шляхом зміни елементів доменної моделі GDM, інших моделей (об'єктної, взаємодії й ін.), додавання нових вимог і понять, а також фіксації їх у моделі характеристик і в конфігурації системи;
- розробка інфраструктури КПВ – опис, збереження, пошук, оцінювання й об'єднання готових КПВ, що розміщаються в репозиторію системи;
- фіксація залежностей між характеристиками моделі, яка позбавляє розробників від деяких конфігураційних операцій, що виконуються, як правило, вручну;
- створення репозиторію КПВ і компонентів багаторазового використання в класі завдань ПрО;
- забезпечення безпеки, захисту даних, змін, тощо;
- забезпечення синхронізації і взаємодії КПВ.

Таким чином, при генерації моделі ПрО для сімейства ПС використовується модель характеристик і набір компонентів для реалізації завдань ПрО. Використовуючи дану модель, знання про конфігурацію і специфікацію домену в мові DSL і компонентів в МП, що беруть участь у

цьому процесі, можна автоматизовано згенерувати окремий член сімейства, а також ПЗ для систем сімейства ПрО.

3.3. Роль діаграм характеристик в специфікації моделей доменів.

В поданому підрозділі розглядається підхід до побудови Предметно-Орієнтованої Мови (Domain Specific Language – DSL), котра базується на концепції характеристичної моделі домена, ключовим елементом якої є діаграма характеристик.

Підхід базується на спробі текстової формалізації графічного подання діаграм характеристик – визначенням правил обробки формалізованих описів. Набір цих правил визначає алгебру операцій діаграм характеристик та призначений для нормалізації цих діаграм і обчислення варіабельності їх конфігурації з урахуванням обмежень.

Описи характеристик мають бути безпосередньо відображені в UML-діаграми, які, у свою чергу, можуть бути використані для генерації на МП, зокрема Java.

Мова DSL становить мову програмування або мову специфікації і використовується для генерації членів сімейства систем у домені застосувань.

Передумову розробки DSL становить детальний аналіз та структурування домену застосувань. Серед існуючих методологій доменного аналізу найбільш відомі ODM (Organization Domain Modeling), FODA (Feature-Oriented Domain Analysis) – характеристико-орієнтований аналіз домену та DSSA (Domain-Specific Software Architectures).

Найбільш важливим результатом доменного аналізу є характеристична модель, яка забезпечує узагальнення і розбіжності членів сімейства домена шляхом індикації тих характеристик, які є загальними для усіх членів сімейства, та тих, що є відмінними.

Ключовим елементом характеристичної моделі є діаграма характеристик, що подає собою графічну нотацію для опису залежностей між характеристиками. Концепція діаграми характеристик успадкована з методу FODA. Вона стисло описує усі можливі конфігурації, що визначаються екземплярами або зразками (instances) програмної системи, виділяючи характеристики, які можуть бути відмінними у кожній з конфігурацій.

На даний час склалась наступна ситуація:

- нотація діаграми характеристик описана лише поверхово та пояснюється здебільшого через приклади. Саме тому визріла необхідність формалізації подання діаграми характеристик шляхом розробки мови DSL для визначення характеристик під назвою FDL (Feature Definition Language – Мови Опису Характеристик понять домену), разом з набором формально визначених операція для обробки виразів FDL;
- діаграми характеристик майже не використовуються на практиці і тому важно знайти реальні приклади діаграм характеристик, застосовуємих у конкретних проектах;
- незрозуміло, яким чином діяти з вже отриманою діаграмою характеристик навіть у вигляді FDL-опису. Тут розглянути у першому наближенні шляхи до відображення FDL-опису діаграми характеристик у діаграму класів UML з тим, щоб створити базу для побудови інструментарію, який має реалізувати поданий підхід, тобто мову FDL.

3.3.1 Мова опису діаграми характеристик.

Діаграми характеристик подають собою лише картинки, що описують системні характеристики. Та для створювання автоматизованих інструментів, призначених для побудови діаграм характеристик і обробки останніх, потрібно текстове подання. Воно має не тільки містити інформацію, що існує

у графічній діаграмі, та бути також придатним для автоматизованої обробки. Визначення складається з множини визначень характеристик (feature definitions): ім’я (назву) характеристики наслідує символ ":" та характеристичний вираз (feature expression). Характеристичний вираз, у свою чергу, може складатись з:

- **атомарної характеристики**;
- **композитної характеристики**: іменованої характеристики, визначення якої з’являється десь в іншому місці;
- **необов’язкової характеристики** (optional): характеристичного виразу, що завершується символом "?";
- **обов’язкової характеристики** (mandatory): переліку характеристичних виразів, який замкнений у конструкції all() (круглі дужки, яким передує слово all) – тобто слід задіяти усі члени переліку, який міститься в дужках;
- **альтернативної характеристики** (що реалізує exclusive – вибір): переліку характеристичних виразів, що замкнений у дужкову конструкцію one-of () – тобто має бути задіяний лише один член переліку із замкнених у дужки;
- **невиключаючого набору характеристик** (який зветься “or-features”): переліку характеристичних виразів, який замкнений у дужкову конструкцію more-of(). Це означає, що можна використовувати не тільки безпосередньо члени переліку, а й їх комбінації;
- **значення характеристики за умовчанням:** default = атомарної характеристиці, тобто слово default наслідує яка-небудь атомарна характеристика;
- та інші (що залишається невизначеним на цей момент) характеристики у формі "…". Тобто це вказує, що поданий набір не повністю специфікований.

SDF подає собою формалізм для визначення синтаксису, який можна порівняти в деяких відношеннях з BNF, та він має більш широку галузь дії,

покриваючи визначення як лексичного так і абстрактного синтаксису. SDF створено для розробки нових мов програмування. [3]

3.3.2 Механізми залежності діаграм характеристик.

Діаграма характеристик описує можливі конфігурації системи. Та для отримання робочої системи ці конфігурації слід реалізувати, причому в якості інструментів реалізації поданий підхід пропонує орієнтуватись на використання мов UML та Java.

Запропонований підхід полягає у перетворені характеристик в класи з використанням принципу спадкування. Результат трансляції FDL в UML може бути поданий шляхом використання XMI – формату обміну інформацією XML Meta data (the XML Meta data Information exchange format). XML-документи можуть бути імпортовані в інструменти моделювання UML такі як Rational, TogetherJ та ArgoUML, які, в свою чергу, можуть використовувати отримані діаграми для генерації, наприклад, класів Java.

На додаток до того, специфікація FDL може бути використана для генерації "редактора конфігурації". Такий редактор подає собою інтерфейс користувача панелі, за допомогою якого розробник продукту може обирати, які характеристики включати.

Наступний логічний крок полягає у створенні предметно-орієнтованої мови (DSL), яка б базувалась на визначенні FDL.

Можна помітити, що оператор у FDL-визначенні близькі до операторів, скажімо, BNF або SDF. Тому визначення FDL може легко стати базисом для граматики мови, призначеної для побудови систем основної прикладної галузі.

3.4. Переваги мови DSL.

Мова DSL в порівнянні з мовами програмування має наступні переваги:

- DSL дозволяє висловити рішення в термінах предметної області на відповідному рівні абстракції. Отже, фахівці в даній предметній області можуть розбирати, верифікувати, змінювати і розробляти DSL-програми. Що дозволить змістити акцент у програмуванні в бік фахівців у конкретній галузі, тобто не фахівців в програмуванні. Це дозволить уникнути втрати інформації або її спотворення при передачі від спеціаліст доменної області програмісту;
- DSL-програми лаконічні, багато в чому самодокументуються і можуть повторно використовуватися для різних цілей. Використання в конструкціях мови понять предметної область дозволить на далі читати програму так же легко, ніби вона була викладена розмовною мовою;
- DSL підвищує ефективність, надійність, переносимість і якість супроводження. Операції, що здійснюються на рівні моделей, набагато ефективніші, набагато легші і підтверджені меншій кількості помилок, ніж операції, що здійснюються на рівна програмного коду;
- DSL містить знання про предметну область, забезпечуючи таким чином можливість їх зберігання і повторного використання;
- DSL дозволяє проводити валідацію та оптимізацію на рівні абстракції, відповідної предметної області. Валідація моделі, використовуючи предикати на кожному рівні абстракції, дозволяє уникнути виснажливого тестування на різних етапах розробки;
- DSL покращує тестування ПЗ і дозволяє автоматизувати цей процес. Замість розробки сценаріїв тестування та ручного набору даних, наприклад, ми можемо згенерувати і застосувати їх на основі декларативної інформації, зафіксованої в моделях. Можна також конфігурувати і контролювати тести та спостерігати за їхнім виконанням, використовуючи моделі. Виконуючи налагодження на рівні моделей, користувач може контролювати систему, застосовуючи для цього вирази на мові моделювання;

- моделі, описані за допомогою одного DSL, можуть бути трансформовані в моделі описані за допомогою іншого DSL. Це дозволяє вільно інтегрувати між собою різні частини системи, написані на різних DSL;
- предметна область може бути описана одному рівні абстракції, а потім перетворена з додатковою деталізацією на більш низькі рівні абстракції, що дозволяє доповнювати модель на різних етапах розробки;
- можливість автоматизувати реструктуризацію, збірку і розгортання. Реструктуризація може бути автоматизованою, використовуючи високоточні доменно-специфічні моделі, налаштовані спеціально на мову програмування. Моделі можуть містити інформацію про збірку, включаючи артефакти, що приймають участь в розробці програмних систем, а також їхні залежності. Вони також можуть містити інформацію про розгортання, включаючи конфігурацію розгортуваних виконуваних програм, обсяги і конфігурації апаратних і програмних ресурсів, необхідних виконуваними програмами.

РОЗДІЛ IV. РЕАЛІЗАЦІЯ ПРОЦЕСІВ ЖИТТЄВОГО ЦИКЛУ ЗАСОБАМИ DSL

Для реалізації DSL в середовищі MS.Net розроблено інструментальний засіб DSL Tools. В состав індивідуального дизайнера цього засобу входить графічний редактор, що реалізовує конкретну доменну модель, зокрема модель ЖЦ [7]. Дизайнер реалізації опису DSL потребує вибору базового шаблону, що містить:

- простий шаблон, який включає тільки два об’єктних поняття і нотацію;
- діаграми класів – шаблон ідентичний діаграмам класів UML;
- діаграми варіантів використання – шаблон, що демонструє діаграми варіантів використання UML. [6]

При побудові нового дизайнера користувач може контролювати вигляд майбутнього DSL. Форми, іконки і властивості компонентів можуть бути налаштовані або додані нові. Всі змінені ресурси будуть включені в дизайнер при компіляції (рис. 4.1.).

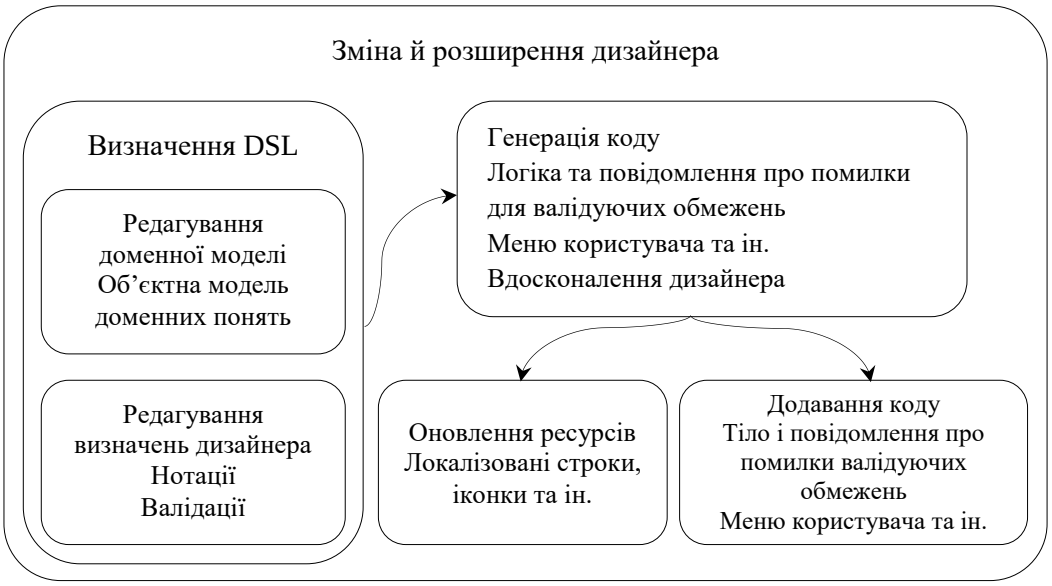

Рис. 4.1. Процес створення нового дизайнера в MS DSL.

Після створення доменної моделі і дизайнера можуть бути додані шаблони коду для вдосконалення і налаштування DSL. Ці шаблони будуть

використовуватися для генерації коду з екземплярів моделі, які користувач створить. Після додавання всіх аспектів дизайнер компілюється і може бути встановлений на Visual Studio.Net, де може бути використаний для генерації артефактів і шаблонів.

Розробники MPS не додали в середовище розробки можливості створення індивідуального графічного редактора, замість якого запропонували інший вид редактора. Цей редактор мов використовує свою власну мову Editor Language. Поле редактора розділене на прямокутні комірки, які можуть містити ключові слова, дужки, роздільники і т. д. Кожна комірка може містити в собі інші комірки. Їх використання має ряд цікавих переваг. По-перше, це дозволяє редактору імітувати текстовий редактор, працюючи на пряму з графовою структурою програми. По-друге, в комірки можна вводити не тільки текст, але й графіки, таблиці і т. д. До того ж коміркова структура це не обов'язковий атрибут, програміст може створити щось своє. Таким чином, Editor Language дозволяє створити коміркову структуру для кожного поняття у мові. Можна визначити які поняття постійні, наприклад, як дужки, а які змінні. Editor Language дозволяє добавити такі можливості, як рефакторинг, автозаповнення, підсвічування синтаксису, підсвічування помилок і т. д.

Графічний DSL в системі Microsoft використовується для реалізації опису домена ЖЦ стандарту в графічного представленні. Дане графічне представлення трансформується до текстового. Далі в розділі 4.1. подається опис процесів ЖЦ: головних процесів, процесів підтримки і організаційних процесів. В Додатку подається текстове подання вказаних процесів мовою XML, що були отримані в результаті звернення до засобу DSL Tools Microsoft.

4.1. Графічне представлення процесів життєвого циклу засобами DSL.

Процеси ЖЦ представлені в цьому розділі включають: головні процеси, процеси підтримки та організаційні. Для домену ЖЦ наведений опис його процесів в графічному представленні.

4.1.1. Головні процеси.

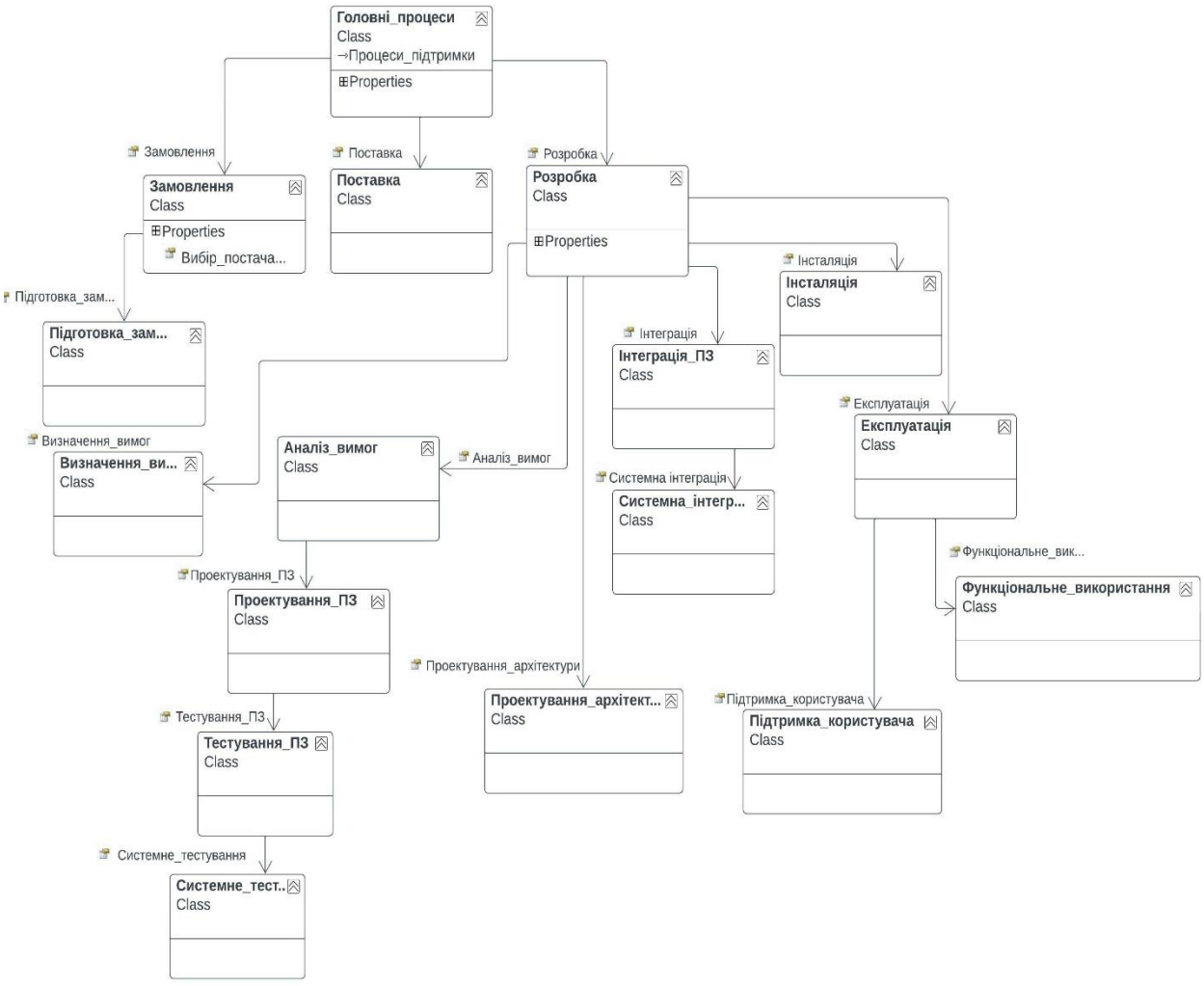

Рис. 4.2. Графічне подання головних процесів.

4.1.2. Процеси підтримки.

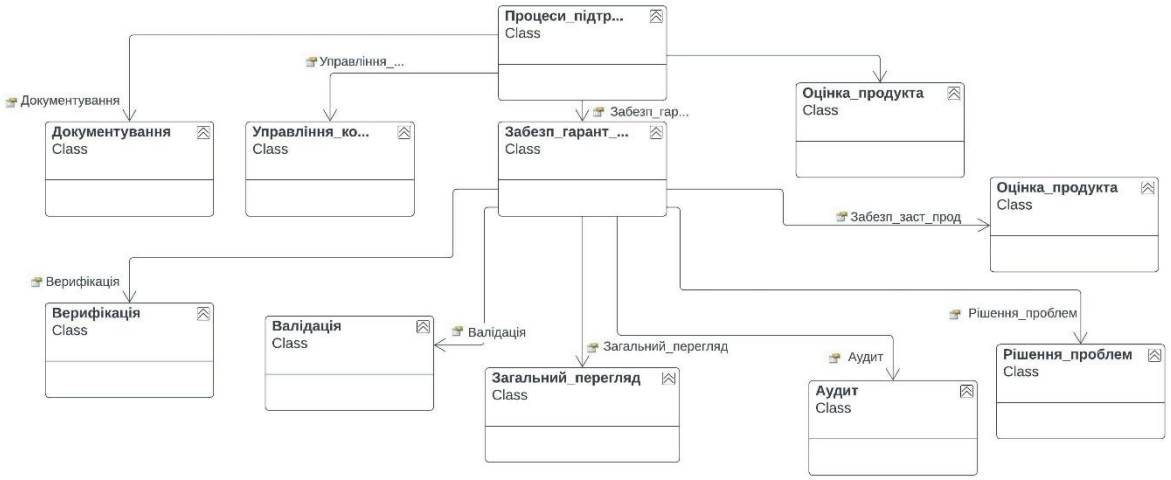

Рис. 4.3. Графічне подання процесів підтримки ЖЦ стандарту.

4.1.3. Організаційні процеси.

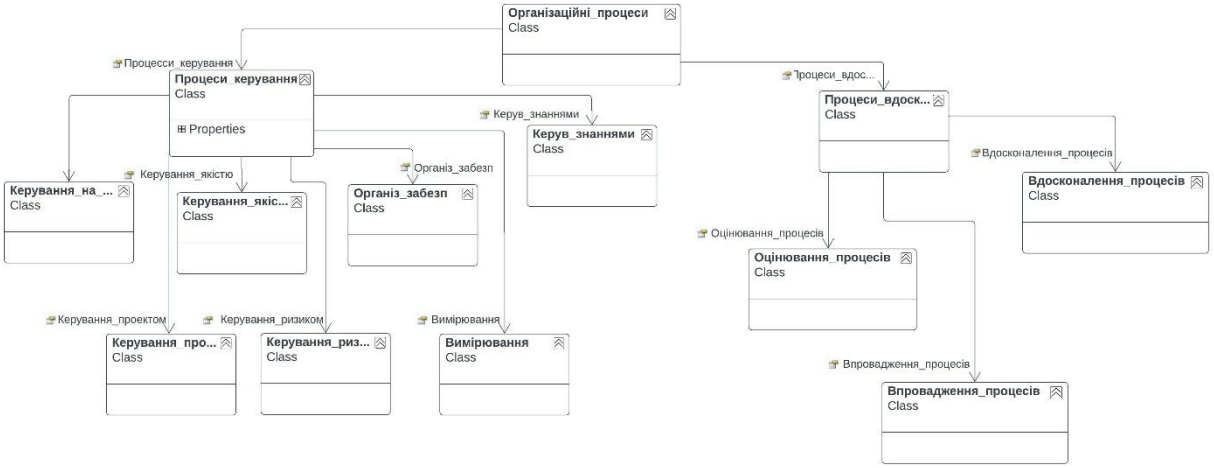

Рис. 4.4. Подання організаційних процесів в графічному представленні DSL.

ВИСНОВКИ

В дипломній роботі проведено дослідження сучасних підходів до представлення предметних областей або доменів, які створюються для відображення різних задач доменів.

Було розглянуто онтологічний підхід до подання моделей предметних областей. Сутність даного підходу міститься у поданні понять предметної області і залежності, які вони мають між собою.

В роботі також розглядається домен програмної інженерії – життєвий цикл стандарту ISO/IEC 12207. Виділено три базових процесів: головні, організаційні та процеси підтримки. Для кожного з них визначені види діяльності (дії), задачі, сукупність результатів (виходів) діяльності і розв’язання задач, а також деякі специфічні вимоги. У стандарті наведено перелік робіт для трьох базових процесів, але не спосіб їхнього виконання і не форма подання результатів.

В стандарті 12207 до основних процесів належать процеси придбання, постачання, розроблення, експлуатації та супроводження. До організаційних належать: керування проектом (менеджмент розробки), якістю, ризиками, тощо.

Мова DSL є продовженням мови UML у напрямку опису понять доменів, їхніх характеристик в графічному представленні з використання класів та їх взаємовідношень.

Приведені моделі характеристик доменів та операції над поняттями доменів та реалізація графічних описів процесів домену ЖЦ на інструментальному засобі MS DSL Tools. Проведена реалізація описів процесів окремо для кожного процесу, тестування і трансформація описів мовою XML.

Таким чином в роботі були досліджені одні з новітніх підходів до подання доменів і виконана реалізація домену ЖЦ в графічному вигляді за допомогою мови DSL.

На мою думку, основними недоліками мови DSL є:

- висока вартість проектування, реалізації та супроводження;
- необхідність у початковому навчанні користувачів;
- відсутність доступних варіантів реалізацій різних варіантів DSL;
- складність визначення області дії DSL;
- необхідність збереження рівноваги між конструкціями, специфічними для предметної області і конструкціями мов програмування загального призначення;
- можливе зниження ефективності порівняно з ПЗ написаним на мовах програмування загального призначення.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Лавріщева К.М. ПРОГРАМНА ІНЖЕНЕРІЯ.–Підручник.– 2008.– 319 с.
2. A Language Workbench in Action – MPS. Martin Fowler.2005, 48k, http://martinflower.com/articles/mpsAgree/html
3. Feature Model Diagrams in text and HTML, 2001. http://www.boost.org/more/feature_model_diagrams.html
4. Arie van Deursen and Paul Klint. Domain-Specific Language Design Requires Feature Descriptions. In *Journal of Computing and Information Technology* – CIT 10, 2002, 1, 1-17.
5. Решетников Е. О. Инструментальное средство проектирования на основе Microsoft Domain-Specific Language Tools-СПб 2008.–114 с.
6. *Jeff Grey et al.*OOPSLA'02 Workshop on Domain-Specific Visual Languages, 2002. http://www.cis.uab.edu/info/OOPSLA-DSVL2
7. *Walkthrough.* Domain-Specific Language (DSL) Tools, 2005.
8. Брауде, Э. Технология разработки программного обеспечения / пер. с англ. –СПб.: 2004. –655 с.
9. Орлов, С. Технологии разработки программного обеспечения / С. А. Орлов. –СПб.:2002. –464 с.
10. Ларионов А. Визуальный язык программирования для Microsoft Visual Studio 2005. СПбГУ ИТМО. –2006. –237 с.

ДОДАТКИ

Графічне подання процесів в розділі 4.1. було використане для їх подання на засіб DSL Tools. Помилки в графічному опису, що були знайдені дизайнером, виправлені за допомогою редактора DSL Tools. Після виправлення отриманий результат кожного процесу, в графічному вигляді DSL, був поданий в мові XML. Далі наведені результати програми для кожного процесу.

Додаток А. Головні процеси

<?xmI version="1.0" encoding="utf-8"?>

<ClassDiagram MajorVersion="1" MinorVersion="1",

<Class

<Position X="3.25" Y="0.5" Width="1.5" />

<Compartments>

<Compartment Name="Properties" Collapsed="true"

</Compartments>

<AssociationLine Name = "Замовлення" Type="Main.Замовлення" FixedFromPoint="true" FixedToPoint="true">

<Path>

<Point X="3.25" Y="1.062" />

<point x="2.125” Y="1.062" />

<Point x="2.125" Y="2" />

</Path>

</AssociationLine>

<AssociationLine Name="Поставка" Type="Main.Поставка" FixedFromPoint="true">

<Path>

<Point x="4.062" Y="1.541" />

<Point x="4.962" Y="2.25” />
</Path>
</AssociationLine>
<AssociationLine Name="Розробка" Type="Main.Розробка"
ManuallyRouted="true"
FixedFromPoint="true" FixedToPoint=-"true">
<Path>
<Point x="4.75” Y="1" />
<Point X="6.312" Y="1" />
<Point X="6.312" Y="2.25" />
</Path>
</AssociationLine>
<TypeIdentifier>
<HashCode>AAAAABACAAAAAAAAAAAAAAAAAAAAAAAE
AAQAAAAA=</HashCode>
<FileName>ГоловніПроцеси.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name= "Замовлення" />
<Property Name="Поставка" />
<Property Name="Розробка" />
</ShowAsAssociation>
</Class>
<Class Name-"Main.Замовлення">
<Position X="1.5" Y="2" Width="1.5" />
<AssociationLine Name = "Підготовка_замовлення"
Type="Main.Підготовка_замовлення"
FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="1.5" Y="2.688" />

<Point X="1.25" Y="2.688" />
<Point X="1.25" Y="3.5" />
</Path>
</AssociationLine>
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAA
AAACAAAA=</HashCode>
<FileName>Замовлення.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Підготовка_замовлення" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Поставка">
<Position X="3.25" Y="2.25" width="1.5” />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Поставка.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Розробка">
<Position X="5" Y="2.25" width="1.5" />
<Compartments>
<Compartment Name="Properties" Collapsed="true" />
</Compartments>
<AssociationLine Name="Визначення_вимог"
Type="Main.Визначення_вимог"
ManuallyRouted="true" FixedFromPoint="true" FixedToPoint=true">
<Path>

<point X="5" Y="2.875" />
<Point X="4.877" Y="2.875" />
<Point X="4.877" Y="3.876" />
<Point X="2.491" Y="3.876" />
<Point X="2.491" Y="5.094" />
<Point x="2" Y="5.094" />
</Path>
<MemberNameLabel ManuallyPlaced="true" >
<Position X="-1.044" Y="0.464" />
</MemberNameLabel>
</AssociationLine>
<AssociationLine Name="Інтеграція_ПЗ" Type="Main.Інтеграція_ПЗ" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="6.5" Y="3.062" />
<Point X="6.75" Y="3. 062" />
<Point X="6.75" Y="3.5" />
</Path>
</AssociationLine>
<AssociationLine Name="Інсталяція" Type="Main.Інсталяція" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint-"true" >
<Path>
<Point X="6.5" Y="2.75" />
<Point X="8.25" Y="2.75" />
<point x="8.25” Y="3" />
</Path>
</AssociationLine>
<AssociationLine Name="Експлуатація" Type="Main.Експлуатація" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>

<Point X="6.5" Y="2.5" />
<Point X="9.312" Y="2.5" />
<Point X="9.312" Y="4.25" />
</Path>
</AssociationLine>
<TypeIdentifier>
<HashCode>AAAAAgAAgAAAAAAAAAAAAIACAAAAAAAAAAAEA
AAAAAEQ=</HashCode>
<FileName>Розробка.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Визначення_вимог" />
<Property Name="Проектування_архітектури" />
<Property Name="Інтеграція_ПЗ" />
<Property Name="Інсталяція" />
<Property Name="Аналіз_вимог" />
<Property Name="Експлуатація" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Експлуатація">
<Position X="8.25" Y="4.25" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAABAAAAAAAAAAAAAAACAAAAA
AAAAAAAA=</HashCode>
<FileName>Експлуатація.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name ="Підтримка_користувача" />
<Property Name ="Функціональне_використання" />
</ShowAsAssociation>

```
</Class>
<Class Name="Main.Підготовка_замовлення"
<Position X="0.5" Y="3.5" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Підготовка_замовлення.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Функціональне_використання" >
<Position X="10" Y="5.25" Width="2.75" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Функціональне_використання.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Підтримка_користувача" >
<Position X="7.25" Y="6.25" Width="2.75" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Підтримка_користувача.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Визначення_вимог"
<Position X="0.5" Y="4.75" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
```

<FileName>Визначення_вимог.CS</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Аналіз_вимог" />
<Position X="3" Y="4" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAI=</Hashcode>
<FileName>Аналіз_вимог.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Проектування_ПЗ" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Проектування_архітектури">
<Position X="4.5" Y="6.5" Width="2.25" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Проектування_архітектури.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Проектування_ПЗ">
<Position X="2.75" Y="5.25" Width="2.25" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAA=</HashCode>
<FileName>Проектування_ПЗ.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>

<Property Name="Тестування_ПЗ" />
</ ShowAsAssociation>
</Class>
<Class Name= "Main.Інтеграція_ПЗ">
<Position X="6" Y="3.5" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Інтеграція_ПЗ.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Системна_інтеграція" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Тестування_ПЗ">
<Position X="2.5" Y="6.5" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAABAAAA=</HashCode>
<FileName>Тестування_ПЗ.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name ="Системне_тестування" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Системна_інтеграція">
<Position X="6" Y="4.75" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>

<FileName>Системна_інтеграція.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Системне_тестування">
<Position X="2" Y="7.5" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Системне_тестування.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Інсталяція">
<Position X="7.5" Y="3" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Інсталяція.cs</FileName>
</TypeIdentifier>
</Class>
<Font Name="Segoe UI" Size="9" />
</ClassDiagram>

Додаток Б. Процеси підтримки

<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
<Class Name="Main.Процеси_підтримки">
<Position X="4" Y="0.5" width="1.5" />
<AssociationLine Name="Документування" Type="Main.Документування" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="4" Y="0.625" />
<Point X="1.25" Y="0.625" />
<Point X="1.25" Y="1.5" />
</Path>
</AssociationLine>
<AssociationLine Name="Управління_конфігурацією" Type="Main.Управління_конфігурацією" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="4" Y="0.938" />
<Point X="3" Y="0.938" />
<Point X="3" Y="1.5" />
</Path>
<MemberNameLabel ManuallyPlaced="true" ManuallySized="true">
<Position X="-0.773" Y="0.579" Height="0.182" Width="1.139" />
</MemberNameLabel>
</AssociationLine>
<AssociationLine Name="Забезп_гарант_якості" Type="Main.Забезп_гарант_якості" >
<MemberNameLabel ManuallyPlaced="true">

<Position X="-1.575" Y="0.068" />
</MemberNameLabel>
</AssociationLine>
<AssociationLine Name="Оцінка_продукта" Type="Main.Оцінка_продукта" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="5.5" Y="1.062"/>
<Point X="7.562" Y="1.062" />
<Point X="7.562" Y="1.25" />
</Path>
<MemberNameLabel ManuallyPlaced="true">
<Position X="-0.432" Y="1.602" />
</MemberNameLabel>
</AssociationLine>
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAABAAAAAAAAAAIAAAAAAAQACAA=</HashCode>
<FileName>Процеси_підтримки.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Документування" />
<Property Name="Управління_конфігурацією" />
<Property Name="Забезп_гарант_якості" />
<Property Name="Оцінка_продукта" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Документування">
<Position X="0.5" Y="1.5" width="1.5" />
<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Документування.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Управління_конфігурацією">
<Position X="2.5" Y="1.5" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Управління_конфігурацією.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Забезп_гарант_якості">
<Position X="4.25" Y="1.5" Width="1.5" />
<AssociationLine Name="Верифікація" Type="Main.Верифікація" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="4.25" Y="2.062" />
<Point X="4.062" Y="2.062" />
<Point X="4.062" Y="2.555" />
<Point X="1.375" Y="2.555" />
<Point x="1.375" Y="2.75" />
</Path>
</AssociationLine>
<AssociationLine Name="Валідація" Type="Main.Валідація" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="4.312" Y="2.368" />
<Point X="4.312" Y="2.607" />

<Point X="4.312" Y="2.607" />
<Point X="4.312" Y="3.094" />
<Point X="3.75" Y="3.094" />
</Path>
</AssociationLine>
<AssociationLine Name="Аудит" Type="Main.Аудит" FixedFromPoint="true" FixedToPoint="true" >
<Path>
<Point X="5.375" Y="2.368" />
<Point X="5.375" Y="2.875" />
<Point x="7" Y="2.875" />
<Point x="7" Y="3.25" />
</Path>
</AssociationLine>
<AssociationLine Name="Рішення_проблем" Type="Main.Рішення_проблем" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true" >
<Path>
<Point x="5.75" Y="2.368” />
<Point X="6.625" Y="2.368" />
<Point x="6.625" Y="2.68" />
<Point X="8.75" Y="2.68" />
<Point x="8.75" Y="3.25” />
</Path>
</AssociationLine>
<AssociationLine Name="Забезп_заст_прод" Type="Main.Забезп_заст_прод" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="5.75" Y="2.062" />

<Point X="6.771" Y="2.062" />
<Point X="6.771" Y="2.524" />
<point x="9" Y="2.524" />
</Path>
</AssociationLine>
<TypeIdentifier>
<HashCode>QAAAAAAAAAAAAAAAQAAABCAAAIAAAAAAQAAA
AAAAAAA=</HashCode>
<FileName>Забезп_гарант_якості.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Верифікація" />
<Property Name="Валідація" />
<Property Name="Загальний_перегляд" />
<Property Name="Аудит" />
<Property Name="Рішення_проблем" />
<Property Name="Забезп_заст_прод" />
</ShowAsAssociation>
</Class>
<Class Name= "Main.Верифікація">
<position x="0.5" Y="2.75" width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Верифікація.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Оцінка_продукта">
<Position X="7" Y="1.25" width="1.5" />
<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Оцінка_продукта.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Валідація">
<Position X="2.25" Y="2.75" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Валідація.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Загальний_перегляд">
<Position X="4" Y="3.25" Width="2" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Загальний_перегляд.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Аудит">
<Position X="6.25" Y="3.25" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
<FileName>Аудит.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Рішення_проблем">

<Position ×="8" y="3.25" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>

<FileName>Рішення_проблем</FileName>

</TypeIdentifier>

</Class>

<Class Name= "Main.Забезп_заст_прод">

<Position X="8" Y="2" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>

<FileName>Забезп_заст_прод.cs</FileName>

</TypeIdentifier>

</Class>

<Font Name="Segoe UI" Size="9" />

</ClassDiagram>

Додаток В. Організаційні процеси

<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
<Class
<Position X="5.75" Y="0.5" Width="2" />
<AssociationLine Name="Процеси_вдосконалення" Type="Main.Процеси_вдосконалення" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="7.75" Y="1.188" />
<Point X="8.75" Y="1.188" />
<Point X="8.75" Y="1.5" />
</Path>
<MemberNameLabel ManuallyPlaced="true" >
<Position X="-0.161" Y="0.362" />
</MemberNameLabel>
</AssociationLine>
<TypeIdentifier>
<HashCode>AAAQAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAA=</Hashode>
<FileName>Організаційні_процеси.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Процеси_вдосконалення" />
<Property Name="Процеси_керування" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Процеси_керування">
<Position X="2.25" Y="1.25" width="2" />

<Compartments>
<Compartment Name="Properties" Collapsed="true" />
</Compartments>
<AssociationLine Name="Керування_на_рівні_організацій" Type="Main.Керування_на_рівні_організацій" ManuallyRouted="true" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="2.25" V="1.578" />
<Point X="1.375" Y="1.578" />
<Point X="1.375" Y="2.5" />
</Path>
<MemberNameLabel ManuallyPlaced="true">
<Position X="0.997" Y="-0. 182"
</MemberNameLabel>
</AssociationLine>
<AssociationLine Name="Керування_проектом" Type="Main.Керування_проектом" FixedToPoint="true">
<Path>
<Point X="2.25" Y="2.161" />
<Point X="2.25" Y="3.75 />
</Path>
</AssociationLine>
<AssociationLine Name="Організ_забезп" Type= "Main.Організ_забезп" FixedFromPoint="true" FixedToPoint="true" >
<Path>
<Point X="4.25" Y="2.102" />
<Point X="5" Y="2.102" />
<Point X="5" Y="2.5" />
</Path>
</AssociationLine>

<AssociationLine Name="Вимірювання" Type="Main.Вимірювання" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point x="4.25" Y="1.906" />
<Point X="5.875" Y="1.906" />
<Point X="5.875" Y="3.75" />
</Path>
</AssociationLine>
<AssociationLine Name="Керування_ризиком" Type="Main.Керування_ризиком" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="3.938" Y="2.161" />
<Point X="3.938" Y="2.372" />
<Point X="4.122" Y="2.372" />
<Point X="4.122" Y="3.558" />
<Point X="4" Y="3.558" />
<Point x="4” Y="3.75" />
</Path>
</AssociationLine>
<TypeIdentifier>
<HashCode>AAAAAAAQAAEAAAABAAAAAAIAAAAgAAAAAAAAAIAAQEA=</HashCode>
<FileName>Процеси_керування.cs</FileName>
</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Керування_на_рівні_організацій" />
<Property Name="Керування_проектом" />
<Property Name="Керування_якістю" />
<Property Name="Організ_забезп" />

<Property Name="Вимірювання" />
<Property Name="Керув_знаннями" />
<Property Name="Керування_ризиком" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Процеси_вдосконалення">
<Position X="8" Y="1.5" width="1.5" />
<AssociationLine Name="Впровадження_процесів" Type="Main.Впровадження_процесів" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point X="9.031" Y="2.368" />
<Point X="9.031" Y="2.668" />
<Point X="9.5" y="2.668" />
<Point x="9.5" Y="4.25" />
</Path>
</AssociationLine>
<AssociationLine Name="Вдосконалення_процесів" Type="Main.Вдосконалення_процесів" FixedFromPoint="true" FixedToPoint="true">
<Path>
<Point x="9.5" Y="1.812" />
<Point x="10.781" y="1.812" />
<Point x="10.781" Y="3" />
</Path>
</AssociationLine>
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAEAAABAA=</HashCode>
<FileName>Процеси_вдосконалення.cs</FileName>

</TypeIdentifier>
<ShowAsAssociation>
<Property Name="Впровадження_процесів" />
<Property Name="Оцінювання_процесів" />
<Property Name="Вдосконалення_процесів" />
</ShowAsAssociation>
</Class>
<Class Name="Main.Керування_на_рівні_організацій">
<Position X="0.5" Y="2.5" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Керування_на_рівні_організацій.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Керування_проектом">
<Position X="1.25" Y="3.75" Width="1.5"
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Керування_проектом.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name= "Main.Керування_якістю">
<Position X="2.5" Y="2.5" Width="1.5" />
<Typerdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Керування_якістю.cs</FileName>
</TypeIdentifier>

</Class>
<Class Name="Main.Керування_ризиком>
<Position X="3" Y="3.75" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Керування_ризиком.cS</FileName>
</TypeIdentifier>
</Class>
<Class Name="Main.Організ_забезп">
<Position X="4.25" Y="2.5" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>
<FileName>Організ_забезп.cs</FileName>
</TypeIdentifier>
</class>
<Class Name="Main.Вимірювання">
<Position X="5.25" Y="3.75" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAA=</HashCode>
<FileNm>Вимірювання.cs</FileName>
</TypeIdentifier>
</Class>
<Class Name="Керув_знаннями" >
<Position X="6" Y="2" Width="1.5" />
<TypeIdentifier>
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>

<FileName>Керув_знаннями.cs</FileName>

</TypeIdentifier>

</Class>

<Class Name="Main.Впровадження_процесів"

<Position X="8.75" Y="4.25" width="2" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>

<FileName>Впровадження_процесів.cs</FileName>

</TypeIdentifier>

</Class>

<Class Name="Main.Оцінювання_процесів"

<Position X="7" Y="3.25" Width="2"/>

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>

<FileName>Оцінювання_процесів.cs</FileName>

</TypeIdentifier>

</Class>

<Class Name="Main.Вдосконалення_процесів">

<Position X="9.75" Y="3" Width="2.25" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA=</HashCode>

<FileName>Вдосконалення_процесів.cs</FileName>

</TypeIdentifier>

</Class>

<Font Name="Segoe UI" Size="9" />

</ClassDiagram>