МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПЗВО «МІЖНАРОДНИЙ КЛАСИЧНИЙ УНІВЕРСИТЕТ

імені ПИЛИПА ОРЛИКА»

Економіко-технологічний факультет

Кафедра інженерних технологій

Кваліфікаційна робота

на здобуття освітнього ступеня магістра

за освітньою програмою «Комп'ютерна інженерія»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: **«МОДЕЛЮВАННЯ СИСТЕМИ УПРАВЛІННЯ КАСТОМІЗАЦІЄЮ МОДУЛІВ ІТ-ПРОЄКТУ»**

Виконав:

здобувач ІІ курсу, групи КІ -20-24

Тацюк Сергій Аатолійович

Керівник:

к.т.н., доцент кафедри інженерних технологій

Гайша Олександр Олександрович

Миколаїв – 2024

ЗМІСТ

ВСТУП

Наука та техніка користуються величезною популярністю у сучасної молоді. Вони – шлях до кар'єрних можливостей у майбутньому. Заняття технічною діяльністю стимулюють у молодих людей інтерес у цих галузях, допомагають їм опановувати необхідні технології та спонукають до дії всіх студентів, школярів та дошкільнят.

У процесі виконання проекту менеджерам треба оперувати значною кількістю даних, які можуть бути зібрані та організовані за допомогою комп'ютера. До того ж, багато аналітичних можливостей, перерахунок графіка діяльності з урахуванням практичних даних, ресурсний та вартісний аналіз передбачають досить складні для неавтоматизованого розрахунку алгоритми.

Зміна систем управління проектами пройшло через низку етапів. Зі зростанням потужності ПК покращувалась функціональність пристроїв, підвищувалися їх ресурси. Із встановленням стандартів обміну даними між системами, розширенням мережевих та інтернет-технологій відкрилися нові зручні випадки для подальшої зміни систем підтримки процесів управління проектами та їх більш досконалого використання. Самі проекти стають дедалі складнішими, що становить спеціальні вимоги до формування інформаційних технологій управлінню проектами.

Отже, **актуальність роботи** зумовлена потребою моделювання системи управління кастомізацією модулів ІТ-проекту.

Об'єктом дослідження є процес керування кастомізацією модулів ІТ-проекту.

Предмет дослідження - Система управління кастомізацією модулів ІТ-проекту.

Метою роботи є розробка моделі системи управління кастомізації модулів ІТ-проекту, що забезпечує підвищення ефективності управління процесом кастомізації модулів ІТ-проекту.

Завдання дослідження:

1. Проаналізувати сучасні механізми керування кастомізацією модулів ІТ-проекту.
2. Провести огляд та аналіз сучасних типових рішень для управління кастомізацією модулів ІТ-проекту.
3. Виконати порівняльний аналіз та вибір методологічних підходів до моделювання системи управління кастомізацією модулів ІТ-проекту.
4. Розробити концептуальну, логічну та фізичну моделі системи управління кастомізацією модулів ІТ-проекту.
5. Перевірити адекватність реалізованої моделі системи керування кастомізацією модулів ІТ-проекту.

Гіпотеза дослідження: застосування запропонованої системи керування кастомізацією модулів ІТ-проекту дозволить підвищити ефективність процесу керування кастомізацією модулів ІТ-проекту.

Методи дослідження: сучасна концепція управління кастомізацією модулів ІТ-проекту, об'єктно-орієнтований підхід, моделювання системи методології UML (Unified Modeling Language).

Наукова новизна дослідження полягає у розробці нової моделі системи управління кастомізацією модулів ІТ-проекту.

Апробація результатів дослідження: Тацюк С.А., Гайша О.О. Захищений хмарний сервер. Магістерські читання – 2024 : матеріали наук.-практ. конф. (26 березня 2024 р.). Миколаїв, 2024. С. 152-154

Практична значимість полягає у збільшенні ефективності роботи підсистеми керування модулів ІТ-проекту.

Структура роботи: вступ, 3 розділи, висновок, список використаних джерел та додаток.

РОЗДІЛ 1. АНАЛІЗ АКТУАЛЬНИХ СИСТЕМ УПРАВЛІННЯ КАСТОМІЗАЦІЄЮ МОДУЛІВ ІТ-ПРОЕКТУ

1.1. Аналіз проблем ефективності управління кастомізацією модулів ІТ-проєкту

Сьогодні в умовах лютої конкуренції та загальної економічної кризи підприємства всіма прийнятними ним коштами намагаються підняти свій економічний коефіцієнт корисної дії шляхом зниження витрат та підвищення ефективності своїх заощаджень. Насамперед це стосується непрофільних пологів занять, таких як ІТ. Але також ясно, що мінімізація витрат на ІТ не може бути універсальним рішенням труднощів компанії, тому що, без сумніву, саме ІТ є одним серед генераторів конкурентних успіхів компаній в даний час і без якого важко уявити хоч якийсь великий бізнес. Ось чому стає ясно, що необхідно дійсно оптимізувати, але не мінімізувати витрати фірми на ІТ, сподіваючись пройти через важкий період, не втративши своїх положень на ринку. Слід зазначити, що в численних західних холдингах управління якраз ефективністю ІТ-проектів давно стало одним з головних і перших інструментів антикризового керівництва. Таким чином, необхідно насамперед оцінювати вплив ІТ-проектів на бізнес певної компанії.

Зараз існує безліч усіляких методів моніторингу та аналізу ефективності накопичувальної привабливості ІТ-проектів. Загалом можна відібрати три основні групи: фінансові, імовірнісні та якісні методи оцінки. Усі вони спрямовані на оцінку ефективності індивідуальних ІТ-проектів за всілякими критеріями, щоб керівництво підприємства могло приймати розраховані та обдумані висновки щодо їх виконання та фінансування.

Під структурою ІТ-проекту розуміються рішення, які приймаються щодо того, як ІТ-проект найкраще відповідає своїй меті. З боку практичного підходу створення структури ІТ-проекту має на увазі під собою написання чистого коду.

Важливими вимогами до такого коду є максимально зрозумілі залежності та логіка, а також структурна організація файлової системи, в якій розміщуються папки та файли.

Які функції потрібно поміщатися в які плагіни? Як дані рухаються за проектом? Які ролі можна зібрати воєдино та ізолювати? Відкликаючись на подібні пункти, можна у різноманітному сенсі прикидати, як виглядатиме закінчений продукт [15].

Модулі в інформаційній системі дозволяють класифікувати та відстежувати різні аспекти вашого бізнесу, такі як продаж, маркетинг, клієнти, продукти, події тощо. Вони бувають двох видів: стандартні модулі і виготовлені на замовлення або модулі користувача.

Зазвичай інформаційна система дозволяє працювати з одним або декількома стандартними модулями для продажу, маркетингу, підтримки клієнтів та управління запасами. Ці зумовлені модулі поставляються з набором полів за промовчанням та макетами. Можна редагувати більшість аспектів стандартного модуля відповідно до вимог конкретного замовника. Наприклад, якщо в модулі Модуль1 є поле Поле1, і ви вважаєте, що жоден з ваших клієнтів не використовує Поле1, ви можете видалити це поле з макета Модуль1. Так само є багато інших опцій налаштування, що надаються стандартними модулями. Параметри налаштування та їх виключення відрізняються залежно від модуля, який потрібно змінити.

Стандартні, попередньо визначені модулі доступні всім користувачам системи незалежно від версії, на яку вони підписані. Іноді стандартні, визначені модулі, що є в системі, не завжди відповідають усім вимогам замовника. У такому випадку система дозволяє створювати новий модуль, залежно від потреб вашого бізнесу. Наприклад, модулі «Контакти», «Посібник» та «Угоди» не будуть ідеальними для ІС освітнього закладу. «Учні», «Вчителі» та «Батьки» були б більш відповідними модулями для них, і вони можуть створювати ці модулі. Лікарня може створювати нові модулі під назвою «Лікарі», «Пацієнти» та «Медсестри».

Щоб задовольнити ці унікальні бізнес-вимоги, можна створювати власні модулі. Завдяки функціональності модулів користувача в системі можна розробляти нові модулі, використовуючи вбудовані інструменти, які не вимагають навичок програмування. Ці модулі можуть легко інтегруватися з основними модулями стандартної системи і не повинні бути автономними модулями.

В даний час більшість великих і не дуже IT-проектів часто виявляють, що рішення, що розробляються та наявні на ринку, що включають стандартні функціональні можливості, просто не підходять під вимоги самого проекту. Одним із можливих рішень у даному випадку можна вважати переробку або доопрацювання одного із запропонованих рішень під вимоги поточного IT-проекту. І кожен проект повинен визначити для себе архітектуру для майбутнього продукту, яка передбачає під собою можливе розширення функціоналу, не забуваючи при цьому думати про можливі пастки та невдачі, які чатують майже на кожному кроці кожного розробника в процесі створення такого рішення.

У цьому роботі вивчені деякі з можливих підходів до кастомізації IT-проекту.

1. Скористайтися підходом з використанням атрибутів, які мають можливість динамічно змінюватися. До такого підходу можна віднести використання моделі Entity-Attribyte-Value. Таку модель, як і, називають Open Schema. Вона думається бути використаною спільно з типовою реляційною моделлю для встановлення і зберігання нових величин нових ознак (атрибутів) сутностей, що трансформується в часі. При застосуванні цієї моделі всі значення атрибутів головним чином вносяться до однієї таблиці з трьох стовпців. Стовпці здебільшого називають як Entity, Attribute, Value:

— Entity – зберігає посилання на сутність, характеристики якої відображаються. У багатьох випадках є визначником сутності;

— Attribute - зберігає посилання на позначення атрибута;

— Value – теперішнє значення атрибуту.

На рис. 1.1 зображено схему даної моделі «Сутність-Атрибут-Значення».

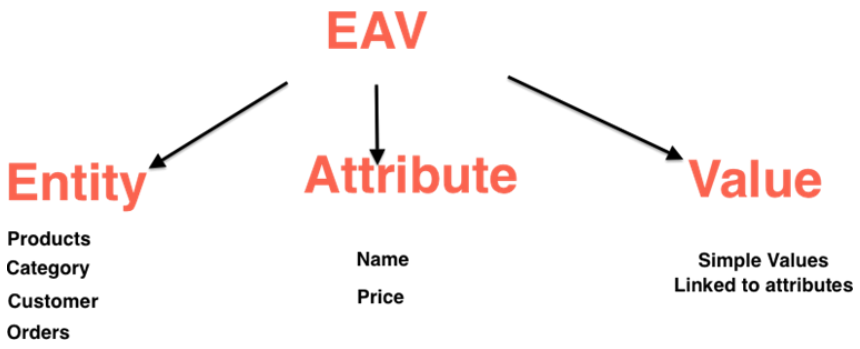

Рис. 1.1 – Модель Entity-Attribute-Value

Визначено, таблиця, яка міститиме опис службових даних, що описують атрибути, має бути обов'язковим складовим схеми. Приклади таких службових даних, що використовуються при описі атрибутів, можуть бути такими:

— опис типу атрибуту;

— обмеження щодо змісту інформації, що включає в себе атрибут;

— посилання на компонент, який використовується в UI для відображення даного атрибуту;

— порядковий номер у черзі на малювання компонента, що містить цей атрибут.

Для використання цієї моделі у проекті потрібно виконати дві речі:

1. Виконати механізм встановлення метаданих. З його сприянням вдасться, наприклад, відзначити, що до сутностей типу "Постійний покупець" додалася чергова властивість "Контактний номер", тип цього рядкового поля, для його показу використовується компонент текстового поля.

2. Створити індикацію та введення вмісту змінних атрибутів на потрібних відеопанелях програмного забезпечення. Конструкція повинна знаходити евентуальний набір атрибутів для наявної сутності в таблиці з оглядом метаданих, рекомендувати компоненти їх поліпшення, а далі з таблиці з даними діставати і відбивати їх значення, відображаючи при згортанні екрана.

Одним із дійсно значущих плюсів у використанні такого підходу можна вважати відсутність потреби у проектуванні та імплементації проекту-розширення. Це означає, що замовник отримує готовий продукт, у якого просто створюються різні динамічні властивості для сутностей у процесі налаштування і навіть у процесі експлуатації.

Щоправда, у такому підході є свої недоліки. Насамперед необхідно сказати про обмеженість використання такого підходу. При виборі моделі динамічних атрибутів має розуміти, що вона дозволить тільки створити додаткові властивості для сутності та відмалювати ці властивості на екрані в заздалегідь відведеному для них місці. При такому підході про зміну властивостей або навіть UI компонентів не може бути мови [23].

Далі необхідно відзначити, що підхід з використанням атрибутів, що динамічно налаштовуються, істотно збільшує навантаження на всю базу даних. Навіть якщо не враховувати зв'язки, якими володіє кожна сутність, процес вивантаження окремого екземпляра будь-якої сутності вимагатиме більше часу та ресурсів, тому що доведеться рахувати не один, а кілька рядків з таблиці. А якщо виникає необхідність у вивантаженні цілої колекції сутностей, прикладом якої можна вважати компонент випадаючого списку для UI, то доведеться зробити N+1 таких запитів, або ускладнювати запити шляхом додавання джойнів, кількість яких дорівнюватиме кількості колонок в таблиці. Все це разом з тим, що в більшості IT-проектів БД є вкрай погано масштабується і однією з найповільніших компонентів ланкою, дуже легко і швидко може призвести до того, що абсолютно вся система перестане функціонувати.

По-третє, адже через структуру бази цілком мудро підтягуватиме вибірки даних для відомостей — замість того, щоб складати наступний по черзі SQL-запит для реляційних даних, стануть необхідними куди більш комплексні запити.

2. Скористатися підходом, який передбачає використання модулів. Використання подібної архітектури дасть можливість спроектувати та імплементувати додаткові функціональні рішення, методи та класи, які можна

буде тримати в ізольованих один від одного та від «ядра» програми модулів. Якщо у замовника виникає потреба в одному з таких модулів, замість того, щоб переписувати проект, достатньо лише написати новий модуль і прописати зв'язки для того, щоб інтегрувати його у свій готовий проект. Щоправда, треба пам'ятати про необхідності спроектувати точки розширення, щоб інтеграція модулів була можлива. Що розуміється під точкою розширення? Зазвичай це спеціальні класи, які займаються тим, що вибирають із уже доступних модулів ті, які мають необхідну в даний момент виконання програми функціональність і у разі виявлення такого модуля передають управління процесом йому. Найпростішими прикладами можуть бути різні кнопки або пункти меню в UI.

На рис. 1.2 показано схему модульної інформаційної системи.

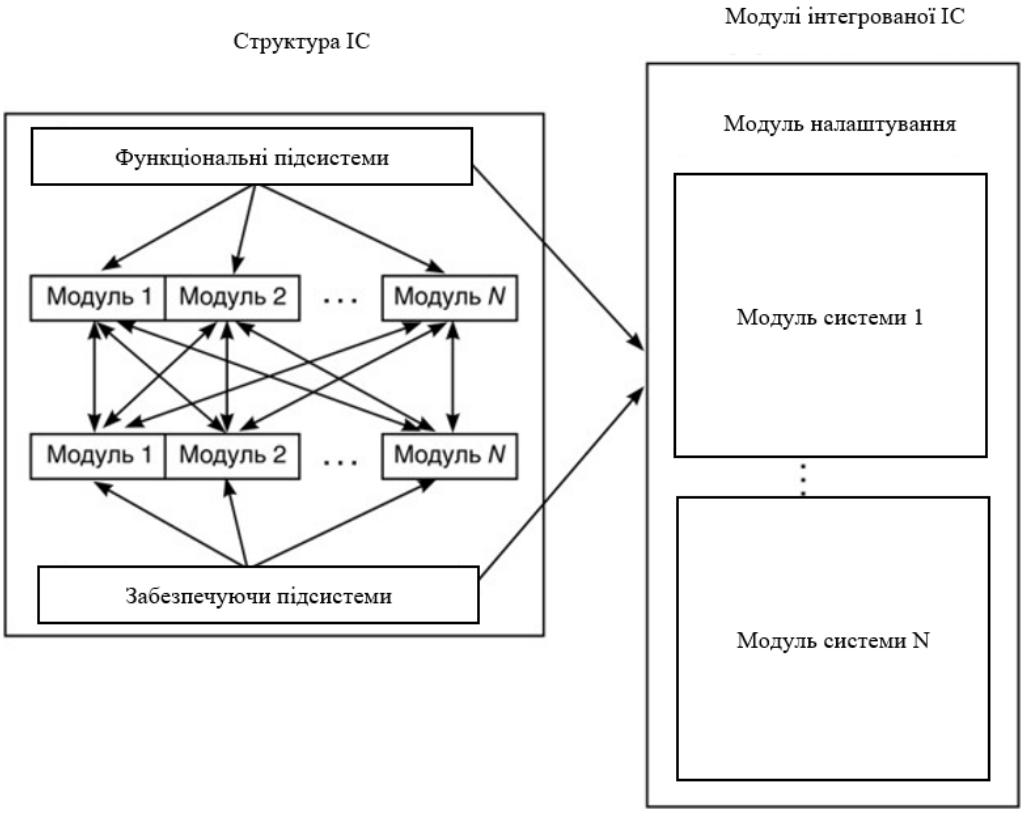

Рис. 1.2 – Модульна інформаційна система

Сценарій логіки та обробників дій у зовнішніх скриптах – часто-густо використовуваний різновид поглиблення функціональності, який і є можливість додати до варіацій плагінів тому, що скрипти викликаються та виконуються конкретними точками програмного забезпечення.

Модульна архітектура потурає зберігати зовсім весь новий функціонал окремо, серед іншого, окремо і від продукту, що без питань є дуже гарною гідністю такого типу архітектури. Абсолютна фізична диференціація продукту та плагінів значно спрощує процедуру оновлення версій програми або модуля, принаймні достатньо лише замістити існуючий компонент свіжим.

Але і в цьому підході є свої підводні камені. Іноді, правда, вони не сильно вплинуть на працездатність продукту, але може статися і так, що це каміння може призвести до неможливості використання модульної архітектури. Насправді використання модулів може бути виправдане лише за умови, що в системі були заздалегідь спроектовані точки розширення. Звичайно, існують такі проектні рішення, де подібні точки практично відсутні і відомі заздалегідь, але в той же час мають місце й інші проекти, в яких доводиться лише гадати, де потрібно поліпшити функціонал. Аналізування подібних ситуацій з виявленням таких місць розширення на майбутнє є процесом непростим і забере багато ресурсів, до того ж воно не дає жодних гарантій успіху. Та й самі точки розширення — це місця коду, які призводять до його ускладнення, що, у свою чергу, призведе до збільшення можливості помилок і складності супроводу. Тож до проектування та імплементації таких точок варто поставитися дуже серйозно [11].

Модульна архітектура дозволяє підприємствам пропонувати різноманітні продукти, щоб швидко реагувати на мінливі вимоги ринку, покращуючи різноманітність продуктів та зменшуючи складність інженерної системи. Модульна технологія, як найефективніша та технологія, що широко використовується, стала найважливішою особливістю проектування структури продукту.

При масовому налаштуванні модулі продукту організовані та керуються відповідно до певних правил, а різні типи модулів збираються для формування модульної організаційної структури. Модульна структура складних продуктів має багато характеристик, таких як різні типи модулів, децентралізовані джерела модулів та незбалансоване використання модулів. Це складна система,

яка впливає та взаємодіє з багатьма видами факторів. Різні фактори призводять до нестабільності модульної організаційної структури, такі як випадкові фактори, включаючи частоту відмов при надходженні зовнішнього модуля та низьку частоту виконання саморобних модулів, а також об'єктивні фактори, включаючи великий попит на модуль та високі показники використання модулів. Ці фактори призведуть до відсутності своєчасного постачання модулів, затримки постачання продукції і т. п., і зрештою позначається на стабільності виробництва. Тому, враховуючи різні фактори, стабільність модульної організаційної структури покращується шляхом зміни типу модуля, покращення структури модуля, розробки нових модулів тощо, отже вивчення еволюції модулів має важливе значення для інженерних додатків [7].

У міру збільшення складності та типу механічних виробів, тип та кількість модулів продуктів швидко збільшуються. Це важливе питання у керівництві еволюції модуля. Організаційна структура модуля для підприємства – це проста статистика кількості різних модулів. Щоб оптимізувати організаційну структуру модулів, необхідно розглянути низку проблем, таких як базові відносини між модулями, кількість модулів та зміни у модулях.

Два важливі міркування при реалізації еволюції та оптимізації організаційної структури модуля такі:

1. Як встановити модель організаційної структури модуля. По-перше, модель модульної структури має бути встановлена для дослідження еволюції модуля. В даний час матриця структурної структури, теорія графів та складна мережа є основними методами моделювання організаційної структури модуля. Методи дослідження описують відносини між компонентами та модулями, встановлюючи модель неспрямованої, спрямованої чи зваженої мережі. Однак більшість об'єктів моделювання залежить від продукту і не враховують організаційну структуру модуля різних продуктів. Було запропоновано використовувати ті ж частини у сімействі продуктів як точку збігу та створити моделі багатопродуктових мереж за допомогою суперпозиції дерева продуктів для розміщення продуктів та компонентів в одній мережі для досліджень, але

існують багаторівневі відносини між продуктами, модулями, компонентами, що не сприяє еволюції дослідницьких модулів.

2. Як забезпечити стабільність організаційної структури модуля щодо еволюції модуля. Еволюція модулів сприяє змінам організаційної структури модулів, а напрям еволюції модулів безпосередньо впливає оптимізацію організаційної структури модулів. Як складна система стабільність організації модуля має вирішальне значення. В галузі дослідження стабільності системи було запропоновано концепцію вразливості енергосистеми та створено слабкий метод аналізу, заснований на перехідній енергетичній функції та штучній нейронній мережі. У 2000 році Альберт Фуад досліджував тендітні джерела, засновані на складній теорії, яка призвела до крихкості системи в нову епоху. Ван запропонував модель грат каскадних відмов, пов'язану з картою, яка забезпечує хороші математичні засоби для моделей каскадних відмов складних мереж. Джин та його команда запропонували крихкість системи за допомогою теорії ентропії та теорії мутацій системи. Крім того, останнім часом з'явилися нові досягнення в теорії крихкості складних мереж та додатках крихкості складних мереж. Однак, порівняно з іншими складними системами, механізм поширення крихкої поведінки, крихкі джерела та інші фактори відрізняються через унікальні характеристики організаційної структури модуля, і відповідний механізм вимагає глибокого аналізу. Крім того, динаміка еволюції модуля визначає характеристики динамічних змін у системі. Стабільність системи як важливий показник оптимізації системи повинна стежити за динамікою оновлення [19].

1.2. Сучасна концепція управління кастомізацією модулів IT- проекту

Почавши написання невеликого, та реального та перспективного проекту, легко на власному досвіді винести переконання, наскільки значно те, щоб програма як добре працювала, так і була добре організована. Не варто

сподіватися, що обґрунтована архітектура потрібна лише величезним проектам (очевидно, що для великих проектів відсутність архітектури може призвести до занепаду). Складність переважно зростає набагато різкіше розмірів програми. Також якщо не потурбуватися про це авансом, то досить швидко настає хвилина, коли вона припиняє бути контрольованою. Хороша архітектура економить напрочуд багато сил, днів і грошей, що нерідко взагалі вказує на те, чи виживе даний проект чи ні за що. І навіть у тому випадку, коли йдеться не більше, ніж про побудову спрощеного проекту, все одно спочатку дуже бажано її спроектувати.

Є можливість сформулювати список у всіх відносинах розумних та всеосяжних критеріїв високоякісної архітектури:

Ефективність системи. Спочатку програма, безумовно, повинна вирішувати призначені завдання і як слід виконувати своє призначення, причому за будь-яких умов. Сюди можна віднести такі параметри, як надійність, безпека, продуктивність, здатність долати збільшення навантаження (масштабованість) тощо.

Гнучкість системи. Будь-який додаток доводиться переглядати з часом — трансформуються вимоги, додаються сучасні. Чим жвавіше і зручніше можна вставити зміни в функціонал, що міститься, чим менше головного болю і помилок це зробить, тим гнучкіше і здатніше скласти конкуренцію система. З цієї причини в рамках розробки слід намагатися зважувати те, що виходить на тему, як це в майбутньому, можливо, доведеться модифікувати. Зміна однієї частини системи не повинна впливати на її інші частини.

Розширюваність системи. Допустимість приписувати в систему чергові сутності та функції, не переступаючи її основної композиції. На початковій стадії в систему є розрахунок закладати лише найважливіший і вимушений функціонал, але водночас архітектура повинна дозволяти легко нарощувати допоміжний функціонал за необхідності. Причому для того, щоб внесення найбільш ймовірних коштувало найменших зусиль.

Масштабованість процесу розробки. Можливість зменшити термін розробки через залучення до проекту нових осіб. Архітектура повинна дозволяти «розпаралелити» процес розробки, щоб ціла низка людей могли діяти над програмою разом.

Тестованість. Код, який простіше тестувати, зберігатиме менше промахів та надійніше виконуватиметься. Але тести не шліфують лише якість коду. Багато розробників приходять до висновку, що вимога

«Висока тестованість» є також ганяючою силою, що автоматично зводить до совість спроектованого дизайну, і відразу одним з найважливіших заходів, що дозволяють розрахувати його якість.

Можливість повторного використання. Систему потрібно конструювати так, щоб її частини можна було використовувати повторно в різних системах. Добре структурований, читаний та зрозумілий код [12].

Супроводжуваність. Над програмним забезпеченням, як правило, працює безліч людей — дехто йде, приходить черговий. Після написання доповнювати програму теж, здебільшого, необхідно людям, які не фігурують у її виробництві. Тому правильна архітектура має давати шанс щодо просто і швидко осягнути систему новій публіці. Програма має бути добре структуралізована, не містити дублювання, нараховувати добре складений код та не заважало б документації. А також вкрай бажано використовувати стандартні, загальновідомі рішення, що часто використовуються, якими користуються програмісти по всьому світу. Чим химерніша система, там складнішим стає її зрозуміти для людей, які цю систему експлуатуватимуть.

Критерії поганої архітектури системи:

1. Жорсткість. Її важко змінити, бо будь-яка зміна впливає надто велику чисельність інших частин інформаційної системи.
2. Крихкість. При внесенні змін несподівано ламаються інші елементи системи.
3. Нерухомість. Код важко використовувати вдруге в іншому проекті, оскільки його надто важко «звільнити» з поточного додатку.

Незважаючи на відсутність одноманітності критеріїв, все-таки основною при розробці великих систем є проблема зниження складності. А для ослаблення складності нічого поза розподілом на елементи поки не винайдено. Комплексна система повинна ґрунтуватися на невеликій кількості більш елементарних підсистем, кожна з яких також будується з компонентів меншого розміру, і т. п., до тих пір, поки найдрібніші фрагменти не будуть досить легкі для безпосереднього усвідомлення і створення.

На рис. 1.3 показано процес створення архітектури інформаційної системи.

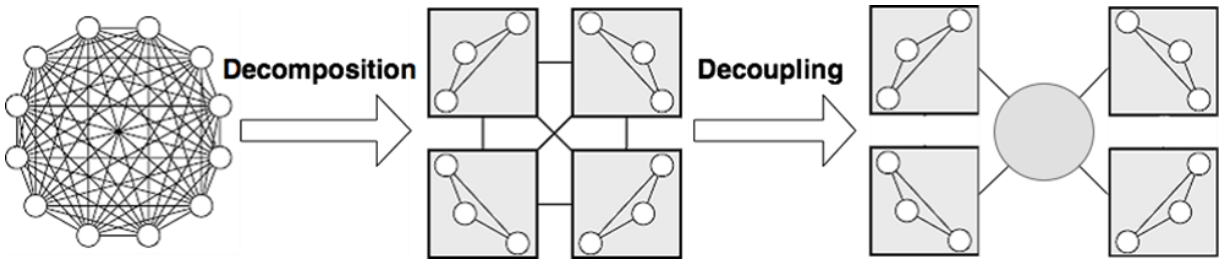

Рис. 1.3 Створення архітектури системи

Надане вище рішення є як єдино відомим, так й багатофункціональним. Крім зниження заплутаності, воно одночасно гарантує гнучкість системи, забезпечує хороші шанси для масштабування, до всього іншого, дозволяє підвищувати надійність завдяки дублюванню критично важливих ланок.

Відповідно, коли розповідається про влаштування архітектури програми, підставі її структури, під тим, головним чином, передбачається декомпозиція програмного забезпечення на підсистеми (функціональні модулі, сервіси, шари, підпрограми) та підготовка їх взаємодії між собою та зовнішнім середовищем. При цьому, чим більш самостійні підсистеми, тим безпечніше сконцентруватися на розробці кожної з них окремо в конкретний часовий проміжок і водночас не стежити за іншими частинами.

За цих обставин програма з погано спроектованої, слабо структурованої та заплутаної трансформується в конструктор, що складається з послідовності модулів/підпрограм, що комунікують між собою за добре відомими та простими

принципами, що, дійсно, і дозволяє перевіряти її складність, крім цього, дає можливість брати всі ті здобутки, які зазвичай порівнюються з поняттям архітектури високої якості:

Масштабованість – здатність системи розширюватись та примножувати власну продуктивність завдяки впровадженню нових модулів;

— ремонтопридатність – при модифікації одного модуля, решті модулів немає необхідності змінюватися;

— замінність модулів – один модуль може бути легко замінений іншим модулем;

— можливість тестування – кожен модуль можна протестувати/змінити/полагодити незалежно від інших модулів;

— перевикористання – модуль можна використовувати в різних проектах та на різному оточенні без будь-яких модифікацій;

— супроводжуваність – програму, розбиту на окремі модулі, набагато легше супроводжувати в міру експлуатації.

Одним словом, метою більшості різних методів проектування є розбиття одного складного завдання на простіші складові [19].

З усього вище сказаного можна дійти невтішного висновку, що прикладом хорошої та якісної архітектури є модульна/блочна архітектура. Для того, щоб спроектувати та отримати на виході якісне архітектурне рішення, необхідно знати про те, як правильно реалізовувати декомпозицію системи. А це, у свою чергу, означає, що дуже потрібно розуміти – яку декомпозицію вважати «правильною» і який спосіб найкраще підходить для її проведення.

1.3. Основа для створення системи керування кастомізацією модулів IT-проєкту

Кастомізація (від англійської to customize – регулювати, змінювати щось, виробляючи відповідне потребам конкретного клієнта) - це індивідуалізація

продукції під замовлення певних споживачів. Це досягається шляхом внесення застосовних або дизайнерських перетворень.

Основна місія кастомізації – створити у покупця відчуття, що продукт робиться саме для нього та вгамовує його особисті запити. Кастомізація вважається найвищою метою взаємодії у сценарії «постачальник матеріальних благ чи послуг – покупець». Вона забезпечує висококонкурентну перевагу завдяки утворенню вищої цінності (переваги) для замовника.

Процес кастомізації впроваджується спеціально на засадах індивідуальних розпоряджень та запитів щодо продукту. Кастомізація проєкту закликає переглянути параметри за бажанням користувача, взяти, кольорову гаму графічного інтерфейсу користувача (серед іншого, і деяких його частин). Це може бути, наприклад, нанесення логотипу замовника на шапку з меню.

Система кастомізації – це чудова психологічна кампанія: поки клієнт комплектує конфігурацію продукту під свої потреби, він починає сприймати себе його власником [8].

Сучасні контури у створенні ІВ наполягають вкладати в архітектуру систем проблематичність динамічного розширення їхнього функціоналу. І, незважаючи на існування помітної кількості відомостей у цій течії, загального врегулювання будови плагінного застосування не існує. Застосування ж зробленого кимось методу дозволу щоразу можливе через специфіки мови програмування чи експлуатованої системи. За аналогією, чужі рішення плагінних систем не завжди доступні для засвоєння, а іноді не в міру хитрі.

Плагіни у різноманітних системах найчастіше мають різними межами функціональності. В ІВ можуть бути висунуті деякі жорстко визначені точки поглиблення - частина функціоналу, що доповнюється сторонніми розробниками. Інакше в повному обсязі система може реалізовувати лише механізм управління модулями, а функціонал весь винесений в особливі плагіни.

На сьогоднішній день одним із відомих і використовуваних підходів до організації додатка, що розширюється, з можливістю кастомізації його функціоналу є модель «сутність-атрибут-значення» (EAV).

Модель «сутність-атрибут-значення» (EAV) – це модель даних для кодування, з ефективним використанням простору, сутностей, у яких кількість атрибутів (властивостей, параметрів), які можуть використовуватися для їх опису, є потенційно величезною, але число, яке буде насправді відноситься до цієї сутності відносно скромне. Такі об'єкти відповідають математичному поняттю розрідженої матриці.

EAV також відомий як модель «об'єкт-атрибут-значення», «модель вертикальної бази даних» та «відкрита схема».

Якщо використовувати цей підхід для представлення даних, то можна помітити суттєві подібності зі зберіганням лише ненульових значень у розрідженій матриці. При використанні моделі даних, що використовує динамічні атрибути, що змінюються, можна помітити, що фактом, який описує сутність, буде пара атрибут-значення. При цьому один рядок у таблиці бази даних, спроектованої за такою моделлю, зберігає один факт. Таблиці EAV часто описуються як «довгі і тонкі»: «довгий» належить до рядків, «тонкий» до кількох стовпців.

Дані записуються у вигляді трьох стовпців:

1. Сутність: описуваний об'єкт.
2. Атрибу та/бо параметр: у більшості випадків його реалізація є зовнішнім ключем, що має відношення до таблиці, в якій цей атрибут визначений. Ця таблиця зазвичай містить такі стовпці, як ім'я параметра, його опис та ідентифікуючий номер, набір будь-яких обмежень змісту для інформації, що зберігається в цьому атрибуті і т. п.
3. Значення атрибут [25].

Далі розглянуто приклад, як можна спробувати подати медичну карту пацієнта в реляційній БД. Зрозуміло, що утворення таблиці (або набору таблиць) із сотнями стовпців важко, оскільки значна більшість стовпців буде

нульовою, оскільки різні пацієнти можуть мати зовсім різний набір захворювань. Щоб ускладнити ситуацію, у багаторічній медичній карті, яка закріплена за пацієнтом протягом часу, може бути низка значень однієї й тієї ж характеристики: зростання та вага, наприклад, змінюються у міру антропоморфних змін у тілі дитини. Нарешті, всесвіт клінічних даних продовжує зростати: наприклад, відкриваються хвороби та втілюються нові лабораторні випробування; це викличе не перестає додавання стовпців і постійний перегляд інтерфейсу користувача. Ситуація, коли список атрибутів часто змінюється, мовою бази даних називається «мінливість атрибутів».

Нижче наведено статичне подання таблиці EAV для клінічних даних, отриманих під час візиту до лікаря з лихоманкою вранці 6 лютого 1999 року. Записи, показані в кутових дужках, є посиланнями на записи інших таблицях, показані тут як текст, а чи не як закодовані значення зовнішнього ключа для простоти розуміння. У прикладі, наведеному вище, значення, які там були використані, були літерними. Однак варто розуміти, що ці значення можуть мати інші формати, які можна заздалегідь зберегти у списку значень. Такі списки особливо корисно створювати у разі, коли стає відомо про те, що деякі значення обмежені (тобто їх можна перерахувати).

Сутність. Для клінічних результатів суб'єктом є пацієнт: зовнішній ключ у таблиці, який містить як мінімум ідентифікатор пацієнта та одну або кілька міток часу (наприклад, початок і кінець дати/часу обстеження), які записуються, коли подія, що описується, відбулася.

Атрибут або параметр. Зовнішній ключ у таблиці визначень атрибутів (у цьому прикладі визначення клінічних результатів). Як мінімум, таблиця визначень атрибутів буде зберігати відповідні стовпці: ідентифікатор атрибута, його ім'я, опис, тип даних, одиниці виміру та стовпці, що сприяють перевіряти введення, наприклад, максимальна довжина рядка і регулярне вираз, максимально і мінімально допустиме значення, набір допустимих значень і т.

Значення атрибуту. Воно залежить від типу даних.

Нижче наведений приклад ілюструє симптоми, які можуть спостерігатися у пацієнта з пневмонією.

(<patient XYZ, 2/6/99 9:30 AM>, <Temperature in degrees Celsius>, "39") (<patient XYZ, 2/6/99 9:30 AM>, <Presence of Cough>, "True")

(<patient XYZ, 2/6/99 9:30 AM>, <Type of Cough>, "З phlegm, yellowish, streaks of blood")

(<XYZ, 2/6/99 9:30 AM>, <Heart Rate in beats per minute>, "98")

Описані вище дані EAV-моделі можна порівняти з вмістом чека з супермаркету (які будуть відображені в таблиці Sales Line Items у базі даних). У чеку вказана лише інформація про фактично куплені товари, замість того, щоб перераховувати всі товари в магазині, які покупець міг купити, але не зробив. Як і клінічні дані для даного пацієнта, товарний чек мізерний [21].

«Сутність» – це ідентифікатор продажу/транзакції, тобто зовнішній ключ у таблиці транзакцій продажу. Вона використовується для внутрішнього маркування кожної позиції, хоча в чеку інформація про продаж з'являється зверху (місце розташування магазину, дата/час продажу) та знизу (загальна вартість продажу).

«Атрибут» – це зовнішній ключ у таблиці продуктів, звідки можна подивитися експлікацію, вартість за екземпляр, бонуси та рекламні знижки і т. п. (Їстівне так само непостійно, як і клінічні дані, можливо, навіть більше того: чергова провізія включається постійно, у той час як інші знімаються з ринку, якщо попит на них низький. Жоден досвідчений творець БД не беззастережно зашифровуватиме деякі продукти харчування, такі, як молоко або пельмені, як стовпці в таблиці.)

«Значення» – це кількість куплених товарів та загальна ціна позиції.

Моделювання рядків, де факти про щось (у разі транзакція продажу) записуються як кількох рядків, а чи не кількох стовпців, є стандартною технікою моделювання даних. Відмінності між моделюванням рядків та EAV-моделюванням (яке можна вважати узагальненням моделювання рядків) наведено нижче:

Строкова таблиця однорідна за фактами, що вона описує: таблиця «Позиції» визначає лише продані товари. Навпаки, таблиця EAV-моделі містить майже будь-який тип факту.

Тип даних стовпця(-ів) значень у таблиці, що моделюється по рядках, заздалегідь визначається характером записуваних ним фактів. У той час, як у таблиці EAV-моделі тип даних значення у конкретній рядку перебуває у владі атрибута у цьому рядку. Саме тому у виробничих системах пряме введення даних у таблицю EAV-моделі може спричинити катастрофу, оскільки сам механізм бази даних не зможе виконати надійну перевірку вхідних даних.

У сховищі клінічних даних моделювання рядків також знаходить численні застосування. Підсхема лабораторних випробувань зазвичай моделюється саме таким чином, тому що результати лабораторних випробувань зазвичай є числовими або можуть бути чисельно закодовані.

РОЗДІЛ 2. МЕТОДОЛОГІЧНІ ОСНОВИ МОДЕЛЮВАННЯ СИСТЕМИ УПРАВЛІННЯ КАСТОМІЗАЦІЄЮ МОДУЛІВ ІТ-ПРОЕКТУ

2.1. Методологія моделювання системи управління кастомізацією модулів

Моделювання – це опис явища дійсності моделлю для накопичення інформації про нього через дослідження з його макетом. Використовуючи поняття «моделювання», зазвичай мають на увазі такий процес планування та імплементації, в результаті якого на виході виходить система, яка максимально точно відображає припущення, що вкладені в планування. На додаток до цього можна сказати, що моделювання – це ще й метод пізнання, в який входять різні компоненти та процеси, прикладом яких можуть бути створення моделей і вивчення моделей.

Моделювання адаптує вивчення предмета покриття його створення, необхідних реконструкції та підстави. Воно мобілізується на дослідження робочої системи, якщо фактичний досвід проводити марно через величезних майнових і трудових витрат, і навіть необхідності здійснення аналізу запланованої системи, інакше кажучи, яка наразі матеріально немає у цьому закладі [2].

У процесі формування моделі дуже часто мають місце такі алгоритми та схеми, як:

— топологічний розподіл технічних засобів та її схема;

— шаблони різноманітних функціональних алгоритмів системи;

— структурна схема системи чи об'єкта;

— схема зв'язку;

— система або об'єкт у вигляді структурно-функціональної схеми та т. п.

Концептуальне моделювання є структурованою процедурою формування систем, що утворюється з наступних етапів:

1. Аналіз (Дослідження).
2. Проектування (дизайн).

3. Кодування (програмування).
4. Тестування (налагодження).
5. Використання (введення в експлуатацію).

Роботу системи у вигляді алгоритмів можна описати, використовуючи одну з форм системного аналізу, що застосовується до великих та складних систем. Такою формою є імітаційне моделювання. Найчастіше така форма застосовується тоді, коли потрібно вивчити те, як функціонують різні процеси залежно та умовами чи коли кількість вхідних даних становить дуже великий обсяг. Одним із її плюсів є те, що імітація може бути припинена на будь-якому з етапів, якщо необхідно провести науковий експеримент та описати його. Надалі результати такого досвіду можна буде використовувати як додаткові вхідні дані для інших етапів процесу імітації, але тільки після їх обробки та оцінки.

Існує ряд методів та принципів розробки ІВ, серед яких є можливість висунути супутні: методи «знизу-вгору» та «зверху-вниз», принципи «дуалізму», багатокомпонентності та інші. Модель, яку створюють за прикладом реального явища, об'єкта або процесу, що містить у собі інформацію про властивості походження або про сам об'єкт, явище або процес, прийнято називати інформаційною моделлю.

Зазвичай зображення текстових інформаційних моделей практикуються природні мови. Поряд із природними мовами (українська, англійська і т. п.) розвинені та застосовуються на практиці формальні мови: системи числення, алгебра висловлювань, мови програмування та інші.

Формальні мови повинні мати зафіксований алфавіт, а правила побудови команд і слів невідступно дотримуються правил синтаксису та граматики. Цим усі формальні мови і відрізняються від природних, у яких таких чітких та обов'язкових до виконання рамок та правил немає.

За допомогою формальних мов описують інформаційні моделі відомого роду формально-логічні моделі.

При опрацюванні нового об'єкта спочатку здебільшого встановлюється його спрощена модель, потім вона стає формальною, інакше кажучи,

позначається з використанням математичних формул, геометричних об'єктів тощо.

Процес побудови всіх інформаційних моделей, у якому використовують формальні мови, називає формалізацією.

Моделі, організовані за допомогою математичних підстав та формул, називають математичними моделями.

Метод «знизу-вгору». Колись усі прийоми роботи не іноземних IT-фахівців було організовано великих електронно-обчислювальних підприємствах чи обчислювальних центрах. Найважливішим завданням у такому роді центрів було здійснення вузькоспрямованих доручень якогось певного установи, але з праці над розробкою творів, що подаються до тиражу. Сучасні начальники майже завжди звертаються до такого підходу, розраховуючи, що куди корисніше і зручніше мати власних професіоналів. Метод виробництва програм «знизу-вгору» сприяє автоматизувати лише деякі робочі процедури, при тому, що вищезазначений метод залучається професійними IT-фахівцями. Даний шлях є як ніколи видатковим, і його намагаються не практикувати сьогодні, саме від нього намагаються відійти малі та середні економічні суб'єкти.

Метод «згори-вниз». Розвиток недержавних та інших сучасних напрямів послужило змістом формування ринку різноманітних програмних засобів. p align="justify"> Інформаційні системи, головними завданнями яких була автоматизація різних аспектів підприємства, наприклад, бухгалтерського обліку або технологічних процесів, розвиваються більш швидкими темпами у зв'язку з попитом. Як результат виходить, що більшість інформаційних систем було створено та впроваджено окремими організаціями з боку або різними командами кваліфікованих програмістів «згори». Це сталося тому, що від такої ІС очікувалося покриття запитів, пов'язаних із різними прикладними областями багатьох користувачів.

Програмісти, які працюють з різними інформаційними базами даних, були різко обмежені можливостями через подібну ідею. Зокрема, вони втратили можливість вибору різних графічних форм, а також у виборі різних

алгоритмічних підходів до розрахунків. Це призводить до висновку, що така ідея перекрила можливість розробки систем, здатних розширюватись відповідно до вимог. Причиною цього стало те, що можливості ІС різко обмежуються жорсткими рамками, що закладаються «зверху», щоб уникнути подібних недоліків у проектуванні та реалізації.

Розв'язання задачі автоматизації організації стало необхідно змінити ідеологію побудови призначених для цього ІВ.

Принципи «дуалізму» та багатокомпонентності. Розвиток суб'єктів господарювання, збільшення чисельності їхніх відділень і клієнтів, поліпшення якості надання допомоги і не тільки зробили серйозні реконструкції в експлуатації та діяльності автоматизованих інформаційних систем, що багато в чому спочивають на пропорційній сукупності двох вищезазначених методів.

На основі аналізу визначень моделі можна символічно відобразити визначення моделі Jdef як носія інформації про оригінал (формула 2.1):

Jdef=<M, N, Z, IS, L>, де (2.1)

— M – оригінал (модельоване явище, об'єкт, джерело інформації);

— N – суб'єкт («спостерігач» по Ешбі), тобто особа, якій знадобилася інформація про оригінал для досягнення своєї мети (дослідження, прийняття рішення тощо);

— Z – ціль або сукупність цілей;

— IS – інфраструктура, що забезпечує моделювання, тобто що включає технології та умови моделювання: TECH = {METH, MEANS, ALG, COND, ...} – сукупність технологій (METH – методи, MEANS – засоби, ALG – алгоритми тощо), що реалізують модель; COND = {φex, φin} – умови існування моделі, тобто. фактори, що впливають на її створення та функціонування (φex – зовнішні, φin – внутрішні);

— L – мова для дослідження гносеологічних аспектів відношення «модель – оригінал» [9].

Вибирати тип моделі можна різними способами, але треба розуміти, що це вибір залежить як цілей моделювання. Властивості вихідних даних,

наприклад, обсяг і характер даних також сильно впливають на цей вибір. Ще одним важливим фактором у виборі типу моделі є обмеження у можливостях самого дослідника.

Отже зібрана модель визначається поняттями, яким їй доводиться задовольняти. У таблиці 2.1 наведено основні засади імітації інформаційних систем управління.

Таблиця 2.1 Основні засади імітації інформаційних систем управління

Принципи імітації	Найменування	Характеристика принципів
	Адекватність	Відповідність моделюваної системи управління цілям та завданням підприємства замовника.
	Узгодженість	Система управління будується на вирішення точного низки задач. Початкові два принципи скорельовані.
	Адаптивність	Система управління має повною мірою підходити всім аспектам активності підприємства замовника.
	Точність	Система управління повинна бути настільки точною, щоб вироблені рішення не виявилися занадто скрутними.
	Збалансованість	Система управління повинна містити всі сфери функціонування, в якій вона застосовуватиметься для виключення неохоплених областей.

	Багатоваріантність	Система повинна мати заготовлені на випадок потреби шляхи реалізації, якщо щось піде не так.

Так як аналіз дозволяє встановити якісь ключові спрямованості діяльності системи, то приступати до формування інформаційної системи управління необхідно на основі цього аналізу, застосованого до вихідних даних, отриманих різних рівнях управління, які або вже відомі, або можуть бути отримані.

2.2. Розробка логічної моделі системи управління кастомізацією

На принципах особливостей процедури врегулювання проектних проблем користувач виробив концептуальну модель системи управління. Модель визначає перелік цілей, необхідний вирішення проблем, пов'язаних з проектною діяльністю.

Діаграму варіантів використання можна використовувати в даному випадку як об'єктну модель системи керування кастомізацією модулів. На такій діаграмі нескладно відобразити основні системні та зв'язки між такими процесами.

Окрім іншого, за допомогою діаграми прецедентів можна відобразити усю структуру згаданої системи. Використовуючи цю діаграму як базу, можна скласти план наступних кроків по проектування системи, керуючою кастомізацією модулів [6].

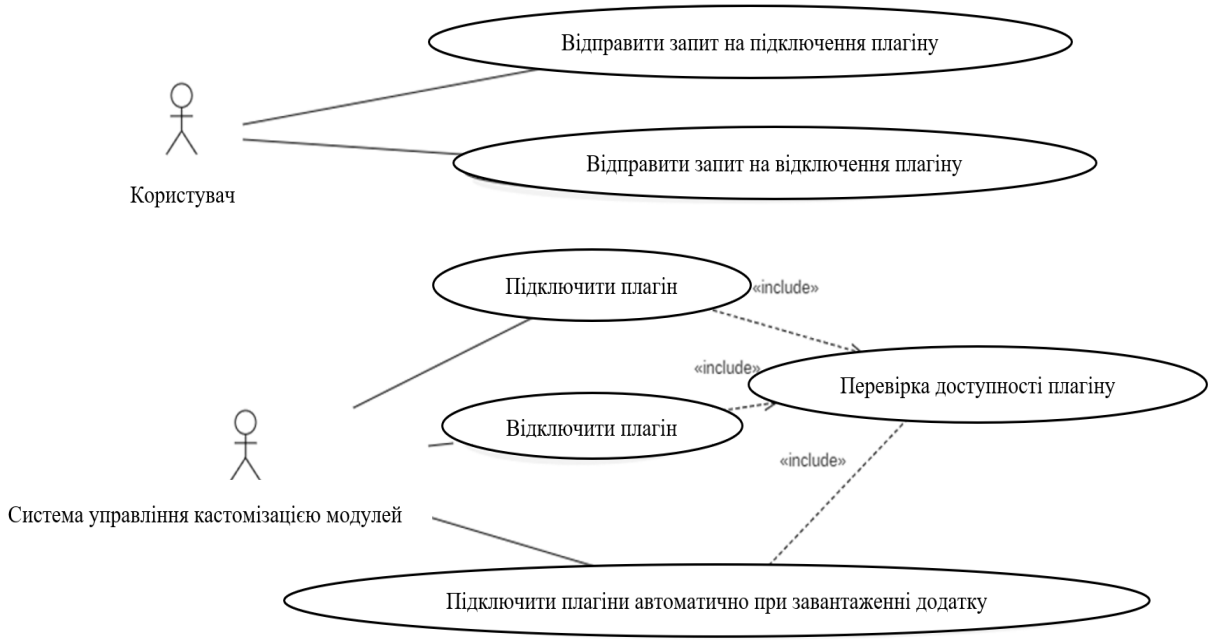

Рис. 2.2 – Діаграма прецедентів системи управління кастомізацією модулів

На діаграмі представлені:

— система управління кастомізацією модулів – блок класів, який є прошарком між користувачем та основними класами, що входять до складу системи управління кастомізацією модулів;

— користувач – особа, яка є співробітником компанії-замовника, відповідальна за налаштування та експлуатацію системи управління кастомізацією.

У таблиці 2.2 наведено коротку характеристику прецедентів проектованої системи управління кастомізацією. У таблицях 2.3 - 2.7 описані специфікації основних прецедентів.

Таблиця 2.2 – Прецеденти

Прецедент	**Характеристика**
Надіслати запит на підключення плагіна	Натискання кнопки «Підключити плагін»
Надіслати запит на відключення плагіна	Натискання кнопки «Вимкнути плагін»

Підключити плагін	Процес підключення плагіна після натискання кнопки «Підключити плагін» або іншої події, що є частиною коду
Вимкнути плагін	Процес відключення плагіна після натискання кнопки «Вимкнути плагін» або іншої події, що є частиною коду
Підключити плагіни автоматично під час завантаження програми	Процес підключення плагінів, що мають властивість автозапуску, старті програми

Таблиця 2.3 – Опис прецеденту: Надіслати запит на підключення плагіна

Прецедент: Надіслати запит на підключення плагіна
ID: 1
Короткий опис: Натискання кнопки «Підключити плагін»
Головні актори: Користувач
Другі актори: Ні
Передумова: Прецедент починається з ініціативи користувача
Основний потік: 1. Користувач натискає кнопку «Підключити плагін»
Постуслів'я: Запит обробляється додатком
Альтернативні потоки: Ні

Таблиця 2.4 – Опис прецеденту: Надіслати запит на відключення плагіна

Прецедент: Надіслати запит на відключення плагіна
ID: 2
Короткий опис: Натискання кнопки «Вимкнути плагін»
Головні актори: Користувач
Другі актори: Ні

Передумова: Прецедент починається з ініціативи користувача
Основний потік: 1. Користувач натискає кнопку «Вимкнути плагін»
Постуслів'я: Запит обробляється додатком
Альтернативні потоки: Ні

Таблиця 2.5 – Опис прецеденту: Підключити плагін

Прецедент: Підключити плагін
ID: 3
Короткий опис: Натискання на кнопку «Підключити плагін» або спрацювання системної події
Головні актори: Додаток
Другі актори: Ні
Передумова: Прецедент починається з ініціативи програми
Основний потік: 1. Додаток обробляє натискання кнопки або системної події 2. Додаток перевіряє доступність плагіна
Постуслів'я: Система діє виходячи з результатів перевірки
Альтернативні потоки: Ні

Таблиця 2.6 – Опис прецеденту: Вимкнути плагін

Прецедент: Вимкнути плагін
ID: 4
Короткий опис: Натискання на кнопку «Вимкнути плагін» або спрацювання системної події
Головні актори: Додаток
Другі актори: Ні
Передумова: Прецедент починається з ініціативи програми
Основний потік: 1. Додаток обробляє натискання кнопки або системної події 2. Додаток перевіряє доступність плагіна

Постуслів'я: Система діє виходячи з результатів перевірки
Альтернативні потоки: Ні

Таблиця 2.7 – Опис прецеденту: Підключити плагін автоматично під час завантаження програми

Прецедент: Підключити плагін автоматично під час завантаження програми
ID: 5
Короткий опис: Підключення плагінів під час старту програми
Головні актори: Додаток
Другі актори: Ні
Передумова: Прецедент починається з ініціативи програми
Основний потік: 1. Додаток перевіряє необхідність автозапуску плагіна 2. Додаток перевіряє доступність плагіна
Постуслів'я: Система діє виходячи з результатів перевірки
Альтернативні потоки: Ні

За допомогою побудованої діаграми варіантів використання можна побачити не тільки межі предметної області, що зазнала моделювання, але й певні вимоги до функцій функцій проектованої системи управління кастомізацією. Реалізація представленої діаграми дозволить описати функції, включені до проектованої системи.

Логічне моделювання – це формування причинно-наслідкових з'єднань між головними факторами, що кваліфікують інформаційні процедури, для відображення цих процедур при дослідженні та оцінці характеристик предметів. Одну із стадій логічного моделювання складає розробка діаграми послідовності.

Принцип, описаний нижче, показує, як відбувається взаємозв'язок між об'єктами та суб'єктами [5; 18]:

1. Система управління кастомізацією модулів ініціює завантаження

доступних плагінів у систему.

2. Клас завантаження модулів завантажує всі доступні модулі, перебувають у папці проекту.

3. Клас завантаження модулів ініціює цикл, в якому для кожного модуля здійснюється перевірка необхідності автопідключення, для чого клас управління модулями подається ім'я модуля.

4. Клас управління модулями здійснює перевірку доступності модуля та перевірку необхідності підключення цього модуля, для чого здійснюються запити до бази даних.

5. Якщо обидві перевірки повернули значення «ІСТИНА», клас управління модулями ініціює відтворення функціонала модуля на графічній формі, надсилаючи необхідну для цього інформацію менеджер форм графічного інтерфейсу.

6. Цикл розпочинає наступну ітерацію. Це відбувається доти, доки всі завантажені плагіни не пройдуть перевірку.

Отже, можна побачити, що перевірку логіки дій для системи було успішно проведено, а також пояснено послідовність роботи системи управління кастомізацією модулів.

За допомогою даної логічної моделі було зроблено опис всіх понять, що використовуються з предметної області і зв'язку між ними. Також можна побачити описані обмеження, накладені на дані, виходячи з обраної предметної області.

Моделювання ІТ-проекту – складний, ресурсномісткий процес, що потребує автоматизації з використанням системи управління кастомізацією ІТ-модулів.

Було виконано функціональне моделювання системи управління, спроектовано логічну модель даних, а також побудовано діаграму прецедентів, діаграму послідовності, діаграму класів та блок-схеми основних алгоритмів роботи для наочного уявлення про модельовану систему управління кастомізацією модулів ІТ-проекту.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА АНАЛІЗ РІШЕННЯ СИСТЕМИ УПРАВЛІННЯ КАСТОМІЗАЦІЄЮ МОДУЛІВ ІТ-ПРОЕКТУ

3.1. Процедура моделювання стандартного ІТ-проекту

У наші дні, коли інформаційні технології все більше входять у повсякденне життя, програмне забезпечення, налаштоване під конкретну

організацію, потрібно практично для будь-якої фірми, чи то великого, чи малого бізнесу. Переважна більшість проектів робиться тільки для конкретного замовника і жодним чином не передбачає кастомізації під інші проекти. А процедура доопрацювання під певний проект займає надто багато часу та трудових та грошових ресурсів.

Щоразу при імплементації проектів впровадження виникають невирішені проблеми:

— збільшення обсягу процесів та процедур;
— зниження швидкості роботи системи;
— неможливість використання системи на повну потужність;
— розтягування процесів застосування на тривалий термін;
— часті зміни вимог замовника тощо.

Майже кожен IT-проект стикається з цими проблемами під час впровадження. Про це відомо вже давно, але точної відповіді на запитання, чому так відбувається, немає досі. Одними з можливих причин можуть бути недостатньо чіткі вихідні вимоги замовника, недостатня якість управління проектом. Підсумовуючи ці причини, на виході отримуємо те, що проект майже не збігається із реальними вимогами. З чого напрошується висновок про те, що під час впровадження проекту якихось важливих аспектів було втрачено.

В умовах жорсткої конкуренції IT-проект можна розглядати як безперечний спосіб покращити управлінські та економічні показники діяльності організації [14].

У будь-якій організації IT-проект варто розглядати з погляду, коли він є невід'ємною частиною великої системи. Причиною такого тісного зв'язку проекту з бізнес-процесами є те, що сьогодні такі проекти потрібні абсолютно скрізь. Але через такий тісний зв'язок розгортання нового чи оновленого старого IT-проекту абсолютно всі згадані процеси торкнуться. Організаційна структура цього підприємства теж не залишиться недоторканою.

На додачу всього, одна зі специфічних характеристик IT-проекту - це наявність модифікацій, яке зачіпає не тільки умови втілення проекту в життя,

але і функцію та/або якісні параметри. Тому до того, як взятися за втілення, формується технічна документація, в якій ясно позначаються призначення проекту, способи його визначення та відповідні параметри, як у форматі технічного завдання. У цьому є необхідність, щоб при проведенні втілення в життя та ліквідації проекту вправі було розібрати по кісточках висновки його здійснення, а мірками такого аналізу прямо і видаються перераховані вище величини і значення. Потім установка, яку можна діагностувати, спрямована на те, щоб уникнути суб'єктивних позицій в аналізі відсотків виконання проекту.
Нижче наведено список особливостей практично будь-якого IT-проекту:

1. Процес впровадження будь-якого IT-проекту в структуру підприємства, що функціонує, завжди має на увазі під собою зміни в цій структурі.

2. У більшості випадків до розробки та впровадження нового IT-проекту залучаються сторонні компанії чи розробники.

3. Впровадження нового IT-проекту завжди пов'язане із ризиками.

4. Дуже часто IT-проект впливає не всі чи майже всі структурні підрозділи підприємства.

Через те, що цих особливостей уникнути неможливо, виникає гостра необхідність застосування різних методів управління проектами в процесі реалізації будь-якого IT-проекту.

Взаємозв'язки між суб'єктами, що здійснюються бізнес-логікою та зафіксовані у низці бізнес-правил, та утворюють діяльність закладу. ІС «відбиває» виразність понять і правила, зводячи і реформуючи інформаційні множини, автоматизує послідовності дій з даними та інформацією та будує підсумки у вигляді пакетів звітних форм. Тому спочатку слід скласти модель компанії, яка бачиться відображенням підприємства та її інформаційно-керуючої системи.

Бізнес-мети будь-якого підприємства у переважній більшості випадків повністю визначають склад більшості можливих компонентів будь-якої моделі та лежать в основі цієї моделі:

— бізнес-функції, що описують те, ЩО робить бізнес;

— те, ЯК виконуються бізнес-функції компанії, що описується різними процесами;

— організаційно-функціональна структура визначає, ДЕ виконуватимуться ті чи інші процеси та функції;

— ролі, які визначають, ХТО буде виконувати функції, а ХТО буде «господарем» процесів;

— фази, з яких буде зрозуміло, КОЛИ впроваджувати всі ці функції;

— Взаємозв'язок між ЯК, ЩО, ХТО, ДЕ і КОЛИ визначається правилами.

Досвід розробки та використання кастомізованих ІС уможливлює умовно виділити наступні основні стадії їх життєвого циклу:

— аналіз вимог до системи визначає те, що має здійснюватись цією системою;

— проектування являє собою різні специфікації підсистем і компонентів та їх взаємозв'язок, необхідні для того, щоб точно сказати, що система повинна проводити;

— розробка – програмне рішення задачі про формування компонентів та підсистем, а також складання підсистем в єдину пов'язану систему;

— тестування – порівняння показників, отриманих після перевірки працездатності системи, з показниками, заданими на етапі оцінки;

— впровадження – процес впровадження системи в експлуатацію;

— функціонування – безпосередньо процес експлуатації, що проводиться відповідно до основних завдань ІВ;

— супровід – забезпечення процесу експлуатації в штатному режимі на підприємстві замовника.

Головна функція системного аналізу становить перетворення будь-яких загальних знань про початковому плані (скажімо, вимог клієнта) у правильно сформульовані позначення та вказівки для кодувальників, серед іншого і

процедура створення функціонального відображення програми. Особливо цей етап береться тим рівнем, на кому ставлять та специфікують:

— не лише внутрішні, але й зовнішні правила роботи програмного забезпечення;

— функціональна характеризування програми;

— порядок поділу функцій між софтом та користувачем та інтерфейси;

— конкретні запити до такого роду частинам системи які інформаційні, програмні та технічні;

— вимоги як до якості, так і до безпеки;

— користувальницька та технічна документація та вимоги щодо її формі;

— експлуатаційні та впроваджувальні умови [7; 10].

Перший етап аналізу – структурний аналіз підприємства - настає з ознайомлення з тим, як складено систему управління підприємством, з розгляду функціональної та інформаційної організації системи управління, формування діючих та ймовірних покупців інформації.

Після аналізу аналітик будує узагальнену логічну модель виходячи з вихідних вимог, що відображають її структуру. У той же час ця модель показує принцип основної діяльності та предметну сферу виконання. На основі цієї моделі аналітиком проектується модель «Як є» (As-Is).

Наступна стадія створення здійснюється із залученням представників клієнта, які знають вимоги до системи. Також можуть бути запрошені незалежні консультанти. Сама стадія здійснюється шляхом аналізу побудованої моделі «Як є». На цій стадії розкриваються вразливі та слабкі ділянки існуючої системи та визначаються шляхи та методи їх зміцнення на підставі вимог замовника до необхідного рівня.

Третьою фазою аналізу, яка повинна охоплювати елементи конструювання, є створення більш повної поширеної логічної моделі, яка виставила б предметну область, що зазнала реорганізації, або деякі елементи цієї галузі, яка так само підходить для автоматизації – модель «Як має бути»

(To-Be) [24].

Четвертою та останньою фазою процедури аналізу є процес розробки «Карти автоматизації». Вона являє собою макет оновленої предметної області і містить чіткі «кордони автоматизації».

У переважній більшості випадків під час проектування системи стадія аналізу вимог передбачає під собою запровадження:

— діаграм бізнес-транзакцій та відповідні їм типи користувачів;

— моделей, що відповідають різним методам, що використовуються в прикладній діяльності, а також списки відповідних завдань;

— класи об'єктів необхідної області та відображають суть інформаційної моделі діаграми, які називаються діаграмами «сутність-зв'язок»;

— топології того, як розташовані користувачі та підрозділи, які обслуговуються цією ІВ;

— параметрів самої системи, захисту даних та параметри інформації.

Наступний пункт – проектування. У цьому положенні проектування – це відшукання і моделювання шляхів розробки, який підходить для додавання функціональності системі методами наявних технологій у світлі зазначених початкових даних та норм. Проектування ІС щоразу починається з формулювання призначення проекту. Основне завдання будь-якого успішного ІТ-проекту визначається тим, щоб до початку прийому системи та протягом усіх днів її застосування можна було забезпечити:

— необхідну працездатність системи та величину адаптації до принципів її роботи, що змінюються;

— продуктивність ІВ і мінімальний час реакції ІВ на запит;

— безперебійну експлуатацію ІС в необхідному режимі, компетентність та відкритість ІС для обробки запитів користувачів;

— недвозначність використання та супроводу ІВ; потрібну ІТ-безпеку даних та права доступу користувачів.

Проектування будь-якої інформаційної системи зазвичай охоплює три

області:

— область, пов'язана з проектуванням структури даних, які будуть реалізовуватись у базі даних;

— область, пов'язана з розробкою програм, графічних форм та різних звітів для здійснення звернень до бази даних;

— область, пов'язана з моделюванням середовища. До цього відносить проектування архітектури, топології мереж тощо.

За результатами проведеного системного аналізу відбувається перехід до стадії проміжного проекту, де покращуються:

— проект впровадження програмно-апаратного оточення, а також проекти, пов'язані із роботою користувачів у системі;

— специфікації мережі та архітектура системи;

— діаграми, що описують потоки даних, присутніх у даній моделі;

— програмне забезпечення системи та блок-схеми прикладного забезпечення (перші — згідно з допущеними моделями середовища ІВ та профілями стандартів) [29].

Стадія детального проектування має на увазі під собою розробку:

— проекту, в якому буде реалізовано середовище ІС та комплекс її функціональних програм;

— даних;

— засобів, призначених для ведення БД, а також структури вхідних в склад ІС мережевих адрес, протоколів обміну інформацією та інших різноманітних складових;

— правил, спрямованих на диференціювання доступу для користувачів, крім цього, засобів втілення цих правил.

Розробка та тестування як усієї системи, так і її окремих компонентів входять до стадії впровадження інформаційної системи.

На фазі використання функціональності системи та її технічної підтримки передбачається управління функціонуванням вищезазначеної ІС, проставлення необхідних трансформацій у БД поточної системи в рамках створення та

покращення призначення самої системи, яке виконується за допомогою прикладних ІТ-фахівців, які використовують додаткові засоби, які вмонтовані в систему.

Етапи розробки моделі ІТ-проекту, які були розглянуті вище, показують, що це дуже складний, ресурсозатратний та тривалий процес, який потребує підвищення ефективності. Отже, має місце розробка моделі системи управління кастомізацією модулів ІТ-проекту, що дозволить скоротити час і ресурси на проектування та імплементацію зв'язків між сутностями, а також на доопрацювання та перероблення коду після внесення змін до вимог замовника.

3.2. Розробка програмного забезпечення системи управління кастомізацією модулів ІТ-проєкту

Процес розробки програмного забезпечення був організований, враховуючи всі раніше побудовані діаграми. Далі наведено основні екрани системи керування кастомізацією модулів ІТ-проекту.

Основним екраном є головне вікно системи управління кастомізацією модулів ІТ-проекту, що з'являється при запуску системи

Звертаючись до цього вікна, користувач може налаштувати підключені модулі, перенаправитися до функціоналу конкретного плагіна, а також переглянути список модулів, що підключаються. До того ж, у цьому вікні можна побачити доступні для підключення плагіни, підключити один або кілька з них, а також відключити будь-який з доданих раніше.

У свою чергу в цьому вікні представлена панель з вибраними операціями для кожного модуля. Використовуючи кнопки на цій панелі, користувач може швидко перейти до функціонала кожного плагіна, що часто використовується.

Нижче наведено функціонал налаштування підключеного модуля на прикладі модуля «Клієнти».

У цьому вікні існують найважливіші елементи, що настроюються модуля системи управління кастомізацією модулів ІТ-проекту Наприклад, обрана

операція, яка рекомендована користувачеві на головному вікні, може бути налаштована шляхом вибору однієї з опцій у випадаючому списку, що містить колекцію основних дій користувача конкретного плагіна. Сам же список, що випадає, може бути поповнений за допомогою кнопки «Додати операцію».

Однією з найголовніших налаштувань, представлених у цьому вікні, є можливість створення кастомізованої таблиці, якщо не влаштовують існуючі або недостатньо функціонали рекомендованих за умовчанням.

3.3. Структурно-функціональна модель системи керування кастомізацією модулів IT-проєкту

Прийнято вважати, що для створення програми, що розширюється, з використанням плагінів потрібно зовсім небагато: описати в classpath файли .jar з плагінами, спроектувати інтерфейс і файли, що містять опис цих плагінів. Після старту системи вона повинна обробити файл з описом та ініціалізувати плагіни.

Насправді ж такий підхід, хоч і простий у проектуванні та імплементації, має кілька великих недоліків.

1. Плагіни обов'язково повинні бути описані в класі програми, а це можливо далеко не завжди.

2. У випадку, коли в classpath потрапляють файли з розширенням .jar, які містять деякі класи, що мають однакові назви, поведінка системи стає непередбачуваною.

3. За такого підходу плагіни мають доступ до об'єктів «ядра» програми, що є небажаним, адже плагіни мають бути ізольовані.

4. Підключення нового плагіна передбачає перезапуск програми, що часто є неприпустимим. Наприклад, якщо розглянути архітектуру клієнт-серверної програми, де такі плагіни виконують роль сервісів, перезавантаження сервера для того, щоб додати такий сервіс є вкрай невиправданим.

Щоб уникнути перерахованих вище недоліків, необхідно імплементувати

такий механізм, який би дав можливість завантажувати необхідні класи під час виконання програми та обмежити їхню «зону видимості». Таким механізмом є ClassLoader'и.

Для ізолювання плагінів один від одного досить просто завантажити їх у різних ClassLoader'ах. Для цього буде створено public інтерфейс, від якого і успадковуватимуться плагіни.

Приклад такого інтерфейсу показаний нижче:

```
public interface PluginInt { public void invoke();
}
```

Так як подібний інтерфейс буде необхідний при створенні будь-якого плагіна в майбутньому, то буде доцільно зібрати цей інтерфейс у файл з розширенням .jar і використовувати як бібліотеку при створенні плагінів.

Для кожного модуля створимо окремий екземпляр класу ClassLoader. Щоб не створювати свій власний ClassLoader, скористаємося готовим URLClassLoader'ом, який чудово підходить для цього завдання.

URLClassLoader — це стандартна бібліотека Java SE 7. Цей завантажувач класів використовується для завантаження класів і ресурсів шляхом пошуку URL-адрес, що відносяться як до файлів .jar, так і до каталогів. Передбачається, що будь-яка URL-адреса, що закінчується символом «/», посилається на каталог. В іншому випадку передбачається, що URL-адреса посилається на файл .jar, який буде відкритий при необхідності.

AccessControlContext (клас AccessControlContext використовується для прийняття рішень про доступ до системних ресурсів на основі контексту, що він інкапсулює) потоку, який створив екземпляр URLClassLoader, буде використовуватися при подальшому завантаженні класів і ресурсів.

Завантажені класи за промовчанням отримують дозвіл лише на доступ до URL-адрес, вказаних під час створення URLClassLoader.

Для створення нового екземпляра цього класу, до його конструктора необхідно подати масив URL папок і файлів .jar і явно вказати об'єкт класу ClassLoader, від якого наш URLClassLoader буде успадковуватися.

Таким чином реалізується ізолювання плагінів один від одного, тому однакові імена класів усередині різних плагінів більше не є проблемою.

Для кожного плагіна зручно використовувати заздалегідь підготовлені властивості, які можна описати у файлі-дескрипторі типу .properties. Ці властивості зручно завантажувати разом із самим плагіном і передавати до класу, який здійснює управління модулями, в якому ці дані (ім'я модуля та його властивості) будуть триматися в колекції типу Map разом з іншими плагінами.

Таким чином, представлене ІТ-рішення має усіма необхідними можливостями для подальшого розвитку та доопрацювання.

ВИСНОВКИ

Метою даної кваліфікаційної роботи є розробка моделі системи управління кастомізації модулів ІТ-проекту, що забезпечує підвищення ефективності управління процесом кастомізації модулів ІТ-проекту.

Впровадження системи управління здебільшого позитивно впливає деякі показники організації, такі, як, наприклад, економічні показники. Це пов'язано з автоматизацією деяких завдань або підвищення ефективності та продуктивності роботи осіб, відповідальних за кастомізацію модулів у будь-якому ІТ-проекті. Вирішенням такого типу питань може бути правильне впровадження системи управління кастомізацією для забезпечення оптимізації трудових та фінансових витрат.

Здійснені в цій магістерській дисертації наукові дослідження демонструють такі основні результати:

1. Було проаналізовано актуальні механізми управління процесом кастомізації модулів ІТ-проекту. Як показав аналіз, для ефективного управління кастомізації модулів використовуються модель Entity-Attribute-Value та архітектура з використанням модулів (плагінів).

2. Проведено аналіз існуючих підходів до моделювання систем управління кастомізацією, який показав, що використання систем управління кастомізацією модулів ІТ-проекту пов'язане з певними складнощами, що зумовлені поганою здійсненністю використовувати одне програмне забезпечення під різні організації.

Тому є актуальністю розробка системи управління кастомізацією модулів ІТ-проекту, що легко адаптується до специфіки проектів конкретних замовників.

3. Проведено порівняльний аналіз методологічних підходів до моделювання розробки системи управління кастомізацією модулів ІТ-проекту. За його підсумками було прийнято рішення використовувати як методологічну основу моделювання системи управління об'єктно-структурний підхід.

4. Проведено функціональне моделювання системи управління, спроектовано логічну модель даних, а також побудовано діаграму прецедентів, діаграму послідовності, діаграму класів та блок-схеми основних алгоритмів роботи для наочного уявлення про моделювану систему управління кастомізацією модулів ІТ-проекту.

З цього випливає, що в даній магістерській дисертації вирішено актуальну науково-дослідну проблему розробки системи управління, яка покликана забезпечити ефективне управління кастомізацією модулів ІТ-проекту.

Найважливішим науковим результатом кваліфікаційної роботи є висновок, що розроблена модель та система управління кастомізацією модулів ІТ-проекту дозволяє збільшити швидкість кастомізації ІТ-проекту.

Таким чином, впровадження в компанії системи управління кастомізацією модулів ІТ-проекту є ефективним та доцільним.

Гіпотеза дослідження підтверджено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ:

1. Азаров О. Д. Основи теорії аналого-цифрового перетворення на основі надлишкових позиційних систем числення : монографія. УНІВЕРСУМ-Вінниця, 2004, 257 с.
2. Бородкіна, І. Л. Web-технології та web-дизайн : застосування мови HTML для створення електронних ресурсів : навч. посіб. Київ : Ліра-К, 2020, 210 с.
3. Бородкіна І. Л. Інженерія програмного забезпечення : навч. посіб. Київ : ЦУЛ, 2019. 204 с.
4. Васюра А.С., Мартинюк Т.Б., Куперштейн Л.М. Методи та засоби нейроподібної обробки для систем керування: монографія. Вінниця: УНІВЕРСУМ-Вінниця, 2008, 175 с.
5. Глибовець М.М., Олецький О.В. Штучний інтелект: підруч. Київ: Академія, 2018. 266 с.
6. Козак Л. І. Основи програмування : навч. посіб. Львів : Новий Світ - 2000, 2019. 325 с.
7. Мартинюк Т. Б., Хом'юк В. В. Методи та засоби паралельних перетворень векторних масивів данних: моногр. Вінниця: УНІВЕРСУМ-Вінниця, 2005. 203 с.
8. Математичні обчислення засобами пакету R-програмування : навч.-метод. посіб. для студ. всіх спеціальностей. Чернігів : ЧНТУ, 2017. 87 с. URL: http://ir.stu.cn.ua/handle/123456789/12520
9. Мельник Р. А. Програмування веб-застосувань (фронт-енд та бек-енд) : навч. посіб. Львів : Вид-во Львівської політехніки, 2018. 247 с.
10. Основи метрології та вимірювальної техніки: підруч. у 2 т. Львів: Видавництво Національного університету «Львівська політехніка», 2005. 345 с.
11. Схемотехніка електронних систем: у 3 кн. кн.1. Аналогова схемотехніка та імпульсні пристрої: підруч. 2-е вид. Київ: Вища школа, 2004. 366 с.

12. Тарасенко В.П., Маламан А.Ю., Черніченко Ю.П., Корнійчук В.І. Надійність комп’ютерних систем. Київ: «Корнійчук», 2007. 256 с.
13. Ткаченко О.М. Об'єктно-орієнтоване програмування мовою Java [Текст] : навч. посібник для студ. напрямів підгот. «Комп'ютерні науки» /Вінницький національний технічний ун-т. Вінниця: ВНТУ, 2006.106 с.
14. Чернега В. Безпровідні локальні комп'ютерні мережі : навч. посіб. Київ : Кондор, 2015. 237 с.
15. Чисельні методи в комп'ютерних науках : навч. посіб. Т. 1. Львів : Новий Світ - 2000, 2019, 469 с.
16. Чисельні методи в комп'ютерних науках : навч. посіб. Т. 2. Львів : Новий Світ - 2000, 2019. 536 с.
17. Шаховська, Н. Б. Системи штучного інтелекту : навч. посіб. Львів : Вид-во Львівської політехніки, 2018, 391 с.
18. Яковлева, О.В. Шматко, Л.В. Гусева, О.О. Паніна. Київ, 2006, 272 с.

Іноземні джерела:

19. A.Czarnecki, L. Klich, C. Orłowski. Simulation of the IT Service and Project Management Environment. Biuletyn WAT, vol. LXII, Nr 1, pp.162 - 165, 2013.
20. A.-G. Dosne, M. Bergstrand, K. Harling, MO Karlsson. Improving оцінка параметрів невизначених розповсюджень в нерівних змішаних ефектах models, використовуючи sampling importance resampling. Journal of Pharmacokinetics and Pharmacodynamics, December 2016, Volume 43, Issue 6, pp 583–596.

Інтернет джерела:

21. Banahan , M. The C Book — Table of Contents [Electronic resource] / Mike Banahan, Declan
22. Brady and Mark Doran. Access mode :https://publications.gbdirect.co.uk//c_book/the_c_book.pdf [in English].
23. Building Back – End Web Apps with Java, JPA and JSF [Electronic resource]. Access mode: https://web-engineering.info/tech/JavaJpaJsf/book/ [in English].
24. Beej’s Guide to C Programming [Electronic resource]. Access mode:

https://beej.us/guide/bgc/pdf/bgc_usl_c_2.pdf [in English].

25. C Elements of Style [Electronic resource]. Access mode: http://www.oualline.com/books.free/style/index.html [in English].
26. Docker for Java Developers [Electronic resource]. Access mode: https://www.oreilly.com/library/view/docker-for-java/978149204262 [in English].
27. Introduction to C Programming [Electronic resource]. Access mode: http://www.tti.unipa.it/~ricrizzo/KS/Data/RMiles/contents.html [in English].
28. Introduction to Programming Using Java [Electronic resource]. Access mode: https://math.hws.edu/javanotes/ [in English].
29. Java Application Development on Linux [Electronic resource]. Access mode: http://javalinuxbook.com/html/downloads.html [in English].
30. Ozdogan M., Sen B., Ozkan S. Digital Transformation Maturity Models: A Systematic Literature Review. Journal of Enterprise Information Management. 2021. 34(6):1521-1547.
31. Bogoviz A. V., Ragulina J. V., Alekseev A. N. Industry 5.0: Theory and Practice of Future Management. Cham: Springer Nature, 2021.
32. Fountaine T., McCarthy B., Saleh T. Building the AI-powered Organization. Harvard Business Review. 2019. Vol. 97, No. 4:62-73.