Київський національний торговельно-економічний університет

Кафедра кібернетики та системного аналізу

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка автоматизованої системи ведення обліку студентів»

Студента 2 курсу, 1м групи,

спеціальності
051 «Економіка»

_____
підпис студента

Мєдвєдєва
Дмитра
Михайловича

спеціалізації
«Економічна кібернетика»

Науковий керівник
доктор економічних наук, професор

_____
підпис керівника

Роскладка Андрій
Анатолійович

Гарант освітньої програми
доктор фізико-математичних наук,
професор

_____
підпис керівника

Гамалій
Володимир
Федорович

Київ 2018

ЗМІСТ

ВСТУП

В сучасному світі автоматизація займає провідне місце серед пріоритетів будь-якого бізнесу чи підприємства. Це відбувається по тій причині, що сама автоматизація хоч і вимагає значних ресурсів на її впровадження, але дозволяє у майбутній перспективі повернути витрачені ресурси і в подальшому економити. Тому кожен бізнес або будь-яка державна установа, яка має численні внутрішні процеси взаємодії розобляють свої або використовують уже готові CRM-системи. Подібний підхід дозволяє за рахунок економії ресурсів ставати більш конкурентоспроможними.

Сучасний підхід до надання освітніх послуг значною мірою нагадує бізнес, як це, власне кажучи і є. Як і у сфері бізнесу, в освітній сфері популярними є рішення з автоматизації адміністративних питань вищих учбових закладів. Таким чином відбувається повна автоматизація внутрішніх адміністративних та навчальних процесів у вищих навчальних закладах переважно за кордоном. Варто зазначити, що такий підхід використовується у більшості розвинених країнах, оскільки навчання за кошти приватних осіб та патенти на наукові розробки дозволяють ЗВО витрачати значні кошти на впровадження, та підтримку подібних систем. У випадку України, освіта не є повністю комерційною, а майже половина студентів навчаються за державні кошти - університети не бажають або не мають змоги витрачати кошти на розробку промислових і впровадження сучасних рішень по автоматизації ЗВО. Подібні ситуації можуть бути спричинені як обмеженим бюджетом, так і застарілим технічним забезпеченням.

З метою скорочення залученого персоналу і пришвидшення прочесу роботи сучасні ЗВО України возробляють власне ПЗ силами викладачів та студентів яке дозволяє у тій чи іншій мірі автоматизувати внутрішні навчальні або (та) адміністративні процесси навчального закладу. Це може бути як, розробка ПО за допомогою мов програмування, так і створення шаблонів документів, таблиць і процесів зберігання інформації за допомого сучасного ПЗ, наприклад пакет

програмного забезпечення від Microsoft.

Іноді це може бути синтез декількох систем або підходів що пов’язані, або не пов’язані між собою, і які слугують для автоматизації як зв’язаних так і незв’язаних між собою процесів, що нерідко призводить до проблем у внутрішній взаємодії, а також створює проблеми у впровадженні нового комплексного ПЗ.

Беручи до уваги сучасну ситуацію, ми можемо прийти до висновку що українському ринку освітніх послуг потрібне массове, добре масштабоване, промислове рішення, яке дозволить уніфікувати та налагодити автоматизацію адміністративних процесів в університетах.

Метою роботи є створення та впровадження автоматизованої системи ведення обліку студентів, спрямованої на збільшення ефективності функціонування закладу вищої освіти.

Для досягнення поставленної мети були сформовані наступні задачі:

1. Дослідити сучасний стан автоматизації ЗВО, методології та підходи до створення і розробки автоматизованих систем.
2. Проаналізувати задачі і цілі створення проекту.
3. Розглянути підходи до створення моделей бізнес-процессів та діаграмм діяльності для подібних проектів.
4. Розробити архітектуру, макет та функціональний прототип проекту.
5. Провести тестування функціонального прототипу проекту.

Об'єкт дослідження: автоматизація внутрішніх процессів закладу вищої освіти.

Предмет дослідження: програмне забезпечення методів управління процессами закладу вищої освіти.

Випускна кваліфікаційна робота складається зі вступу, троьох розділів, висновків, списку використанних джерел. Загальний обсяг роботи складає 131 сторінок.

РОЗДІЛ 1
МЕТОДОЛОГІЯ РОЗРОБКИ ТА АНАЛІЗ СИСТЕМ АВТОМАТИЗАЦІЇ ДІЯЛЬНОСТІ ЗАКЛАДУ ВИЩОЇ ОСВІТИ

1.1. Системи автоматизації та серверні рішення

В сучасному світі автоматизація займає провідне місце серед пріоритетів будь-якого бізнесу чи підприємства. Це відбувається по тій причині, що сама автоматизація хоч і вимагає значних ресурсів на її впровадження, але дозволяє у майбутній перспективі повернути витрачені ресурси і в подальшому економити.

В загальному, на сьогодні, існує два типи подібних систем - виробничі та інформаційні.

Автоматизована виробнича система (АВС) - це організаційно-технічна система, яка складається із способів автоматизації різних видів діяльності людини та робочим персоналом об’єкту, який здійснює виробничу, організаційну та керівну діяльність[12,11].

Автоматизована інформаційна система (АІС) - це система що реалізує інформаційну технологію і підхід до виконання встановлених функцій за допомогою комплексу засобів і методик автоматизації, для використання її в майбутньому персоналом об’єкту[12,11].

В залежності від виду діяльності підприємства, виробництва, компанії або фірми - відрізняють такі види автоматизованих систем як:

- Автоматизовані системи керування (АСК) - які поділяються на підтипи в залежності від виду об’єкту керування:
  - Автоматичні системи керування технологічними процесами (АСК ТП);
  - Автоматичні системи керування підприємствами (АСК П);
  - Автоматичні системи керування виробництвом (АСК В);
- Системи автоматизованого проектування (САП) - які поділяються на підтипи в

залежності від виду об’єктів проектування:

- Системи автоматизованого проектування і розрахунку (САП Р);
- Системи автоматизованого проектування технологічних процесів (САП ТП);

- Автоматизовані системи наукових досліджень (АСНД):
- Автоматизовані системи оброблення та передавання інформації - :
  - Автоматизована інформаційно-пошукова система (АІПС);
  - Автоматизована система інформаційно-термінологічного обслуговуваня (АСІТО);
- Автоматизовані системи технологічної підготовки виробництва
- Автоматизовані системи контролю та випробувань
- Автоматизовані системи, що об’єднують у собі функції вищенаведених систем.

У процесі свого функціонування, за складом і видом структури, автоматизована система є об’єднанням[11,29]:

- Комплексу технологічних засобів автоматизації (ТЗА) - що є сукупністю взаємоузодженних комплексів і компонентів технічного, програмного та інформаційного забезпечень, що виготовляються, розробляються і постачаються як продукція технічного та виробничого призначення:
  - Програмне забезпечення - це набір програм та програмного функціоналу на електронних носіях інформації, а також керівництвом користувача та програмною документацією ;
  - Технічне забезпечення - це набір технічних засобів, безпосередньо необхідних для роботи інформаційної системи, а також належна технічна документація на ці засоби і технологічні процеси;
  - Інформаційне забезпечення - це сукупність загальних правил кодування технічної інформації та єдиної системи класифікації, стандартизованих систем документації і масивів корисної інформації, яка використовується автоматизованою системою керування, в тому числі форм документів, схем,

масивів, графів та протоколи обміну даними.

- Організаційно-методичного забезпечення автоматизованої системи - сукупністю документів, що дозволяють охарактеризувати: організаційну структуру об'єкта та системи автоматизації, які необхідні для визначення конкретних функцій, що автоматизуються; діяльність в умовах впровадження та подальшого функціонування системи, а також форми звітування про результати діяльності;
- Фахівців - персонал об’єкту який використовує вищенаведені компоненти під час виконання своєї професійної діяльності.

Внутрішню будову подібних систем характеризують за допомогою конструкцій, що описують сталі і стійкі зв'язки між їх внутрішніми елементами. При описі автоматизованої системи використовують види структур, що відрізняються типами елементів і їх внутрішніми зв'язками[12,11,41,40]:

- Функціональні - в яких елементами виступають функції, завдання і процедури, між якими є інформаційний зв’язок;
- Технічні - в яких елементами є пристрої, компоненти та комплекси; між якими в якості зв’зку виступають лінії і канали зв'язку;
- Організаційні - в яких елементами є як колективи людей, так і окремі виконавці, між якими є інформаційні зв’язки, та зв’язки підпорядкування і взаємодії;
- Документальні - в яких елементами є неподільні складові частини та документація автоматизованої системи, між якими є зв’язки взаємодії та підпорядкування;
- Алгоритмічні - в яких елементами виступають алгоритми, пов’язані інформаційним типом зв’язку;
- Програмні - в яких елементами є програмні модулі та вироби, між якими є керуючий тип зв’язку;
- Інформаційні - в яких елементами є форми існування і подання інформації в системі, а типом зв’язку виступають операції перетворення інформації в системі.

Елементи автоматизованої системи за своїми ознаками можна поділити на окремі

групи - підсистеми, які у свою чергу, можна розділити на підсистеми ще нижчого рангу[40]. Ділення на підсистеми можна виконувати лише при слідуванні правилам поділу систем на підсистеми: кожна система виконує лише одну неподільну функцію, а зв’язок між окремими підсистемами вводиться лише при наявності істотного зв’язку між відповідними функціями підсистем, усі зв’язки мають бути простими, усі підсистеми одного ієрархічного рівня повинні реалізовувати єдину, загальну функцію системи. Поділ на підсистеми більш низького ієрархічного рівня відбувається доти, поки отримана під час поділу підсистема не перестає містити в собі окремий функціонал, адже це протиречить правилам поділу. У такий спосіб, підсистеми найнижчого рівня реалізують пряме управління окремими знаряддями праці або технологічними процесами. Підсистеми вищого рівня здійснюють керування через підсистеми більш нижчого рівня.

Залежно від структури та розмірів автоматизованої системи керування - подібні підсистеми можна розділити на групи, у такий спосіб розділяючи або об'єднуючи певний функціонал управління в окремі структури.

Реалізація функціональних і прикладних завдань в АСК ведеться через спільний доступ користувачів до бази даних розташованої у внутрішній мережі підприємства, або на віддалених серверах з більш жорсткою системою аутентифікації. Технологічно структура АСК - це глобальна або локальна комп'ютерна мережа, що складається з численних автоматизованих робочих місць (АРМ), обє’днаних спільним доступом до системи.

Кожне АРМ - це сукупність технологічних та програмно-апаратних засобів, що забезпечують взаємодію фахівців різних спеціалістів та фахівців різних профілів і спеціалізацій з АСК, а через мережеві засоби - з всією інформацією розміщеною на серверах компанії або підприємства[11].

Рис 1.1 Складові ERP-систем

Джерело: [42]

Кожне автоматизоване робоче місце в мережі АСК має відповідати переліку правл:

- вчасне задоволення інформаційних вимог користувача;
- короткий час відповіді на запити спеціаліста;
- пристосування до рівня підготовки користувача;
- простота та швидкість освоєння роботи на АРМ;

- надійність і простота обслуговування;
- спроможність роботи через локальну або глобальну мережу.

Більшість подібних мереж забезпечують доступ до необхідної інформації використовуючи архітектуру клієнт-серверних рішень, де клієнтом є АРМ спеціаліста.

Клієнт-серверна архітектура є одним із сучасних архітектурних шаблонів проектупання програмного забезпечення, а також є панівною концепцією при створенні великих розподілених мережевих систем і передбачає інформаційний обмін та взаємодію між ними. Для такого типу архітектури необхідні три компоненти[9,11,12]:

- сервер або набір серверів, які є місцем розташування бази даних, серверної частини системи і надають інформацію іншим програмам або системам, яу відповідь на запити;
- набір програмних інтерфейсів, клієнтів, які використовують сервіси та інформацію, яку отримують від серверів;
- мережа, що забезпечує інформаційну взаємодію між клієнтами та серверами.

Сервери можуть бути фізично та інформаційно незалежними один від одного. Клієнти також функціонують асинхронно і незалежно один від одного. Немає фіксованої належності окремого клієнта до сервера. Звичайною є ситуація коли до одного серверу одночасно надсилаються запити від різних клієнтів, але у той же час, кожен клієнт може звертатися до декількох незалежних серверів. Клієнти мають інформацію про доступні їм сервери, але часто не мають даних про існування та функціонування інших клієнтів[9].

Важливо чітко розуміти, хто або що є клієнтом. Це може бути як фізичний комп'ютер, з якого відбуваються запити до інших комп'ютерів в мережі, так і клієнт-серверне ПЗ , або користувачі, які за допомогою належного програмного та апаратного забезпечення хочуть отримати доступ до тих чи інших данних або сервісів.

Загальноприйнятим є твердження, що клієнти та сервери - є перш за все програмними модулями. Часто вони розміщені на різних комп'ютерах і фізично розташовані в різних локаціях, але бувають ситуації, коли обидві частини програми - і клієнтська, і серверна, фізично розташовані на одному компютері, тоді такий сервер називають локальним.

З середини 90-х років почала розвиватися трирівнева клієнт-серверна архітектура, яка потребує відділення прикладного рівня системи від модуля керування даними. Виділяється окремий програмний рівень, який зосереджує в собі прикладну логіку системи. Побічні програми можуть функціювати під управлінням спеціальних невеликих серверів, хоча запустити подібну програму можливо і під управлінням простого веб-сервера. А керівництво даними і доступ до них здійснюється сервером на якому розміщена система керування базою даних.

Для взаємодії з системою користувачу достатньо використати стандартне ПЗ - веб-браузер. Це позбавляє будь-якої потреби завантажувати та встановлювати спеціально розроблені програми. Але для цього користувач має отримати у своє розпорядження інтерфейс, який дозволить йому повноцінно взаємодіяти з системою[10,9].

Користувач формує запит та надсилає його на сервер, який здійснює обробку. Якщо необхідно сервер звернеться до серверних програмних модулів, які оброблять запит і в разі необхідності звернуться до сервера з базою даних[9]. Сервер із СКБД проводить операції з даними, що знаходяться в системі і утворюють собою її реляційну модель та інформаційну основу. Окрім того, він може здійснити параметризовану вибірку з бази даних відповідно до запиту і, обробивши її, надіслати її модулю проміжного рівня, яким найчастіше є серверна частина системи, для подальшої обробки. Інформація, з яку обробляє сервер даних, часто реалізована як реляційна база даних. В рідких випадках - як нереляційна база даних.

Часто серверні модулі проміжного рівня та веб-сервер фізично розташовані на одній машині, хоч і становлять собою логічно незалежні модулі програми[10].

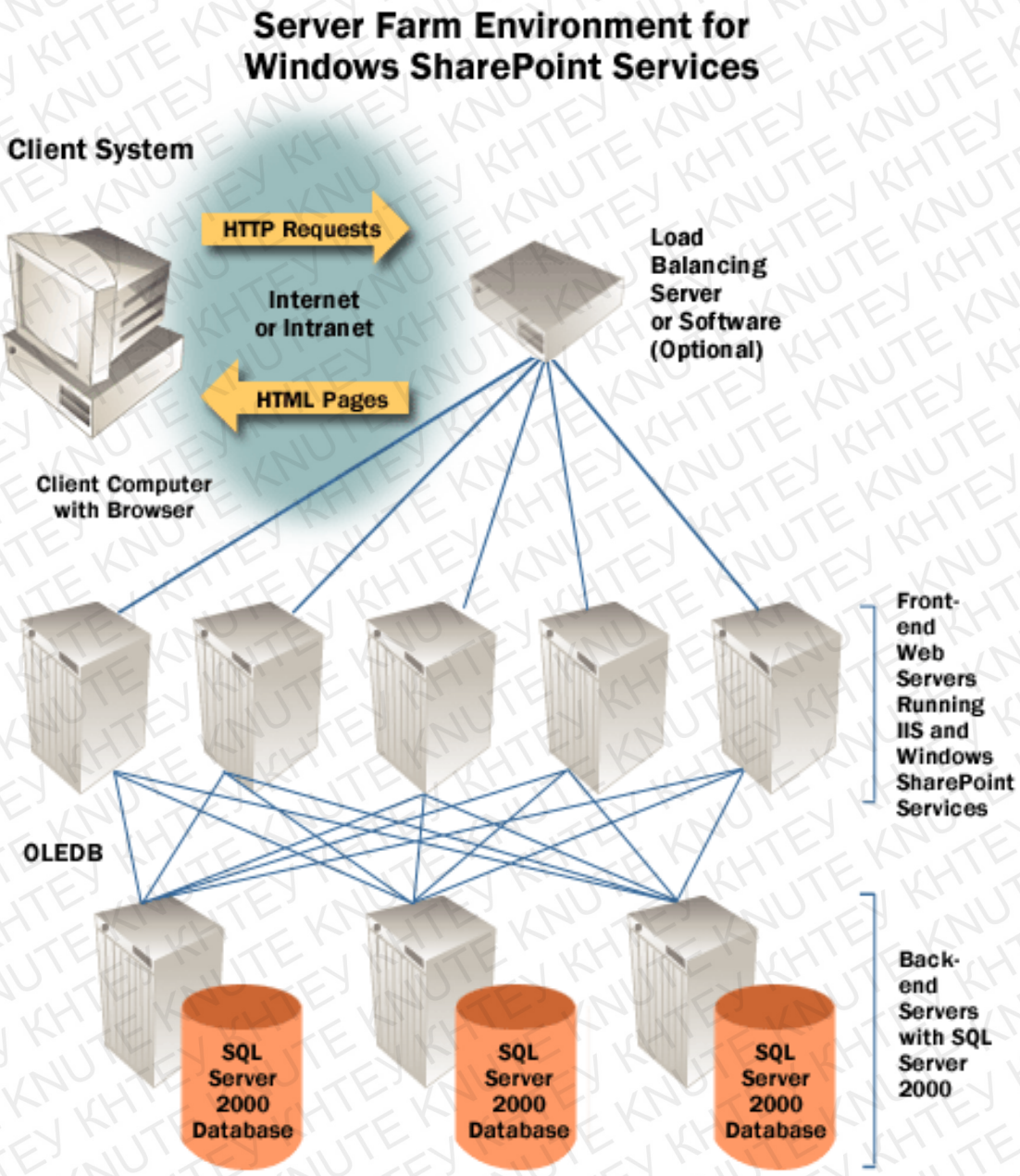

Рис 1.2 Структура клієнт-серверної архітектури БД

Джерело: [43]

Причинами популярності серверних рішень серед великих підприємств і корпорацій є:

- надійність - сервери це технічно-аппаратне забезпечення яке має працювати в режимі 24/7, і відповідно часто комплектується дублюючими елементами, які дозволять забезпечити 99,999% працездатності сервера (час недоступності серверу в

середньому складає до 6 хвилин на рік).

- розміри - сервери встановлюють на стандартизоване обладнання і відповідно всі юніти пріводяться до стандартних розмірів, що спрощує їх промислове виробництво, розміщення та використання.
- ресурси - сервери спеціалізуються в двух протилежних напрямках: нарощування ресурсів, що веде до збільшення місткості та продуктивності серверу, та зменшення ресурсів, що веде до зменшення розмірів серверу та зменшення енергоспоживання.
- масштабованість - можливість нарощувати потужність серверу за рахунок використання більшої кількості процесорів, оперативної пам'яті і т.д., або їх заміни на більш продуктивні.
- розмішення та обслуговування - сервери розміщують у спеціально обліднаних приміщеннях, що називають дата-центром. Існує два шляхи розміщення: створення свого дата-центру, або оренда приміщення у спеціалізованих промислових дата-центрах, в яких може одночасно розміщуватись більше тисяці серверів. Керівництво серверами здійснює кваліфікований спеціаліст - системний адміністратор.

1.2. Засоби розробки системи автоматизації для мультиплатформенного рішення

На сьогоднішній день найуніверсальнішим рішенням для розробки будь-якого програмного забезпечення для массового використання є веб-браузер[10]. Адже більшість браузерів портуються на різні платформи і підтрімуються більшісттю, якщо не всіма, найпоширенішими системами.

Більшість корпорацій притримуються подібної думки розробляючи свої власні системи. Адже подібне рішення має ряд переваг, які дозволяють йому домінувати серед інших:

- Для оновлення системи не потрібно завантажувати файли з інтернету, або

запускати і встановлючати їх на персональному комп’ютері. Розробнику або системному адміністратору достатньо лише оновити продуктовий сервер, і усі користувачі одночасно отримають доступ до нових розробок у межах системи.

- Оскільки користувачу немає потреби завантажувати програмне забезпечення на свій персональний комп’ютер, це одразу підвищує безпеку персональних данних користувача. Менша кількість всановлення ПЗ на машину - менше можливостей зловмисницькому ПЗ отримати доступ до ПК.

- Веб-браузер - це програмне забезпечення, що автоматично встановлюється в більшості системах, тому користувач зможе отримати віддалений доступ до свого робочого місця.

- При продажу ліцензій на використання створеного ПЗ компанія може бути впевнена, що ніхто не скопіює і не зможе використовувати створене ПЗ безкоштовно. Кожен користувач - це окрема ліцензія, а при її закінчені доступ у систему просто відключається.

Окрім перевал подібне рішення має і ряд мінусів, які іноді можуть бути настільки значними, що корпорації приймають пішення випускати десктопне програмне забезпечення[39]:

- Велика кількість користувачів вимагає великої серверної потужності. І, хоча будувати величезні серверні кластери цілком можливо, це вимагає значних витрат на придбання та обслуговування серверного обладнання.

- Зберігання інформації відбувається теж на серверах компанії, і це зобов’язує компаніїї не тільки захищати особисті данні користувачів, але й багаторазово дублювати інформацію, для уникнення її знищення або пропажі.

- Велика кількість користувачів передбачає навність великого відділу операційної підтримки, який буде оперативно вирішувати проблеми користувачів і допомагати їм.

Одною з невід’мних частин будь якого серверного мультиплатформанного ПЗ є база данних[41]. Тобто місце збереження даних, їх опис а також засоби для їх обробки.

Сучасні методології розробки ПЗ взагалі виділяють бази данних в окрему частину программи, і створюють її власну екосистему, яка при побудові моделі ПЗ - знаходиться під серверною частиною. При побудові подібної екосистеми враховуються недоліки обраної СКБД і обирається модель побудови БД. Наприклад може існувати три сервера, один яких буде головним а два інші підлеглими. В такій моделі два підлеглих сервера дублюють інформацію з головного сервера, і один з них виконує роботу з видачі інформації у відповідь на запити. А головний сервер, навпаки, записує данні в таблиці. Данні з головного сервера завантажуються на підлеглі з інтервалом 2-5 хвилин[9]. І у випадку коли головний сервер «падає» його роль на себе бере один з підлеглих, а той що залишився продовжує виконувати роль підлеглого. Якщо компанія може собідозволити, то таких підлеглих серверів може бути декілька, і вони навіть можкть працювати з окремими незалежними програмними модулями системи. Таким чином забезпечується безперервна робота БД для великих веб-орієнтованих додатків[10].

Серед реляційних баз даних варто відзначити три найпоширеніші і найпопулярніші:

- MySQL - це компактний багатопотоковий сервер бази данних, який характеризується швидкісттю, стійкісттю та простотою використання. Вважається гарним рішенням для використання у малих та середніх програмних додатках. Портується на всі основні платформи (Linux, MacOS, Windows)[9].
- Oracle DB - потужний інструмент для зберігання данних від компанії Oracle. Для використання поптребує купівлі ліцензії. Але навіть так, купується багатьма компаніями, корпораціями і підприємствами, тому що має дуже багато інструментів всередині себе, які дозволяють краще керувати і обробляти дані. Наприклад: такий необхідний інструмент як Flashback Query, що дозволяє відновити данні, що були видалені. Або вмонтована мова PL/SQL, яка дозволяє реалізувати логіку роботи бази, майже так само як і звичайної програми[9].
- PostgreSQL - об’єктно-реляційна база данних яка є найближчим і найповнішим

аналогом комерційній базі данних Oracle DB. Є проектом з відкритим кодом, але, незважаючи на це, є широко розповсюдженою, і розробляється самими компаніями, які її використовують у процессі своєї діяльності[9].

Серверною стороною веб-орієнтованого ПЗ є своєрідний контроллер, який дозволяє користувачу ефективно взаємодіяти з даними, що знаходяться всередині БД, і який не потребує від користувача ніяких знань і навичок роботи з БД.

Серверна частина ПЗ може знаходитись на окремо виділенному під його потреби сервері, окремо від баз даних. Ця частина програмного забезпечення може бути реалізована за допомогою більшості існуючих сьогодні мов програмування. Хоча існують мови програмування, які дозволяють взаємодіяти з БД простіше ніж інші. Наприклад об'єктно-орієнтована мова програмування Java, яка має програмно передбачений плагін JDBC (Java Data Base Connector), що дозволяє доволі просто отримувати дані з БД, і реалізовувати звичні SQL операції у вигляді запитів до БД зі сторони серверу[10].

Окрім влаштованого плагіну JDBC, для Java існують спеціально розроблені фреймворки (Hibernate, JPA, OpenJPA, Ebean) які дозволяють використовувати технологію прогрмавання «Об'єктно-реляційна проекція», що в свою чергу дозволяє провести зв'язок між об'єктами[32,34,35], якими оперує серверна частина системи, із таблицями, в яких зберігаються дані в БД[9]. Це дозволяє позбутися багатьох проблем при розробці і проектуванні, але вимагає більш потужних серверів, оскільки подібні ОРМ-рішення є доволі ресурсомістким.

Останньою частиною веб-орієнтованого ПЗ є саме відображення, яке бачить користувач, і з яким безпосередньо взаємодіє[10]. І це фактично є набір веб-сторінок, які через контроллер (серверну частину) взаємодіють з базою даних.

Подібні засоби розробки систем автоматизації ПЗ в цілому, дозволяють побудувати систему, яку можна буде використовувати на будь-якій операційній системі. Інфраструктурно подібну систему можна буде розвернути майже на будь-якій платформі, а для її використання кінцевим користувачем потрібно лише

наявність веб-оглядача.

1.3. Аналіз використання систем автоматизації у ЗВО та оцінка рівня автоматизації ЗВО України

Сьогодні, програмне забезпечення яке використовують заклади вищої освіти, сильно відрізняється від університету до університету. Починаючи звичайним Excel-ем, і закінчуючи самостійно написаним ПЗ. Але навіть при такій розмаїтості усі види програмного забезпечення можна умовно розділити на такі категорії:

- Неліцензовене ПЗ - як це не дивно, але більшість університетів України користуються застарілим, неліцензованим програмним забезпеченням у процессі своєї дільності. Операційні системи (Windows), системи для роботи з електронними таблицями (Excel), для роботи з презентаціями (Power Point), та текстових редакторів (Word) - усе перелічене є, на данний момент, застарілим ПЗ, яке не передбачає автоматизації само по собі, за рахунок застарілості версій. Вирішити цю проблему допомогла б покупка ліцензованих пакетів ПЗ у компанії Microsoft, або використання пакетів программ проприетарного типу (програми з відкритим кодом). Проте економія коштів адміністрацією і звичка персоналу працювати з конкретним пакетом програм не дозволяють провести подібне оновлення систем ПЗ, яке використовують університети.
- Ліцензоване ПЗ (яке передбачає часткову автоматизацію) - деякі ЗВО України купують ліцензію на використання пакета офісних програм компанії Microsoft або використовують пропріетарне ПЗ на умовах відкритої ліцензії. Подібне рішення дозволяє частково автоматизувати внутрішні процесси університету за рахунок використання хмарних технологій та більш досконалих систем внутрішніх коммунікацій.
- Ліцензоване ПЗ (яке передбачає значну автоматизацію автоматизацію) - деякі ЗВО купують спеціальзоване програмне забезпечення, яке розроблено спеціально для

автоматизації внутрішніх організаційних та операційних процесів університету. Воно може складатися з різних модулей, які можна купувати окремо і налаштовувати окремо під кожен ЗВО. Прикладом подібного ПЗ є GS-відомості та Politec Soft.

- Самостійно написане ПЗ - в даному випадку вуз або замовляє розробку спеціалізованого програмного забезпечення під свої потреби у команди незалежних розробників, або реалізує розробку силами внутрішніх технологічних відділів самого університету.

Окремим типом програмного забезпечення - є ПЗ яке поставляється державою, наприклад ЄДЕБО, і є загально-державною електронною базою з питань освіти. Вона реалізує принципи загальнодержавної уніфікації інформації з питань освіти, і дозволяє значно зменшити обсяг паперових документів в обігу як при вступі в ЗВО, так і при подальшому навчанні.

Сьогоднішня тенденція автоматизації ЗВО в Україні не є позитивною, хоча вузи і прикладають деякі зусилля для руху вперед у цьому напряму. Прикладом цього є самостійне створення внутрішніх систем автоматизації, які дозволяють врахувати особливості ведення адміністративних справ в кожному окремому ЗВО. І це дійсно є найбільш ефективним рішенням у сучасних реаліях ведення справ всередині ЗВО.

За кордоном, наприклад у США, системи автоматизації ЗВО займають окреме місце серед систем автоматизації. І оскільки освіта у Сполучених Штатах переважно платна, то університети часто просто купують найбільш функціональне ПЗ, з тих, що представлені ринком, або створюють його самостійно силами внутрішніх технологічних інститутів.

Подібне ПЗ часто складається з таких модулів як[7,28]:

- Студентська інформаційна система управління - модуль керування персональною та операційною інформацією студентів (Повне ім’я, дата та місце народження, стать, документи, національність, освіта і т.д.)
- Система управління людськими ресурсами - модуль керрування персональною та адміністративною інформацією персоналу ЗВО.

- Система адміністративного управлінням - модуль адміністрування що дозволяє формувати і здійснювати управлінські дії спрямовані на підвищеня рівня ефективності діяльності.
- Система обробки результатів - модуль що відповідає за підрахування результатів навчання кожного студента.
- Система управління заробітною платою - модуль керування заробітньою платою працівників ЗВО.
- Система управління фондами
- Система управління бюджетом - модуль розподілу бюджету університету.
- Система управління бухгалтерського обліку - модуль управління надходженнями та видатками університету.
- Система обробки рахунків
- Система управління інвестиціями
- Система управління аудитом
- Система управління нерухомим майном - модуль керівництва всім нерухомим майном університету.
- Система управління студентськими аудиторіями - модуль керівництва та розподілу студентських аудиторій під час навчального процессу.
- Система електронного документообігу - модуль хмарного змерігання документів, та їх взаємне використання.
- Система інвентаризації - модуль інвентаризації обладнання та приладів, які використовуються під час навчального процесу, та знаходяться у власності ЗВО.
- Система адміністрування та керування системою - модуль адміністративного керівництва системою автоматизації.

Таким чином відбувається повна автоматизація внутрішніх адміністративних та навчальних процесів у вищих навчальних закладах в Сполучених Штатах Америки[28]. Варто зазначити, що такий підхід використовується у більшості

розвинених країнах, оскільки навчання за кошти приватних осіб та патенти на наукові розробки дозволяють ЗВО витрачати значні кошти на впровадження, та підтримку подібних систем. У випадку України, освіта не є повністтю комерційною, а майже половина студентів навчаються за державні кошти.

На американському ринку, серед систем автоматиизації ЗВО виділяються такі як: MyEdu – University Automation Software, Eifell Corp Services and Custom eLearning, CyberVision University Management System, і т.д.

Таким чином рівень автоматизації ЗВО України можна вважати помірно низьким, порівняно з університетами США та інших країн, зважаючи на те ПЗ яке вони використовують. І, оскільки, українські ЗВО не вкладають значних коштів у автоматизацію своїх адміністративних та навчальних процессів, то можна очікувати що найближчими роками рівень автоматизації вищих навчальних закладів не буде зазнавати значних змін.

Сьогодні єдиним значним кроком вперед в автоматизації української сфери освіти - є ЄДЕБО, єдина державна база з питань освіти, яка є автоматизованою системою збирання, реєстрації, оброблення, зберігання та захисту відомостей та данних з питань освіти. Власником цієї електронної системи - є держава, розпорядником - Міністерство освіти і науки України, а технічним адміністратором - державне підприємство «Інфоресурс».

З 2012 року ця система забезпечує інформаціїне супроводження вступних кампаній в заклади вищої освіти. З 2014 відбулось часткове розширення функціоналу і заклади освіти почали використовувати дані ЄДЕБО для виготовлення студентських квитвів державного зразка. А з 2015 року дані що зберігаються в ЄДЕБО є підставою для виготовлення документів про вищу та професійну освіту.

З 2018 року єдина державна електронна база з питань освіти представляє собою інтегровану інформаційно-телекомунікаційну систему, технічні засоби якої перебувають в межах території України і складається з комплексу автоматизованих робочих місць, об’єднаних в єдину інформаційну систему засобами зв’язку з

використанням технології віддаленого доступу, має підключення до мереж зв'язку загального користування з розмежуванням прав доступу, забезпечує захист від порушень цілісності інформації, забезпечує різні рівні доступності відкритої інформації та інформації з обмеженим доступом, вимога щодо захисту якої встановлена законом.

З точки розу автоматизованих інформаційних систем ЄДЕБО підходить під усі критерії, але дана система майже ніяк не автоматизує внутрішні процеси ЗВО, які пов'язані з рухом студентів в межах конкретного університету, а виступає більше в ролі незалежного власника інформації.

З іншого боку загальнодержавний характер поширення цієї системи дозволяє відслідковувати і контролювати рух студентів на рівні учбових закладів, і надає функціонал для використання його інтегрованими локальними системами управління процесами, якщо такі існують.

Іншим варіантом вирішення питання недостатньої або повної відсутності автоматизації у ЗВО - є локальне розв'язання подібних задач.

З метою скорочення залученого персонялу і пришвидшення прочесу роботи сучасні ЗВО України возробляють власне ПЗ яке дозволяє у тій чи іншій мірі автоматизувати внутрішні навчальні або (та) адміністративні процесси навчального закладу. Це може бути як, розробка ПО за допомогою мов програмування, так і створення шаблонів документів, таблиць і процесів зберігання інформації за допомого сучасного ПЗ, наприклад пакет програмного забезпечення від Microsoft.

Іноді це може бути комплекс декількох систем або підходів що пов'язані, або не пов'язані між собою, і які слугують для автоматизації як зв'язаних так і незв'язаних між собою процесів, що нерідко призводить до проблем у внутрішній взаємодії, а також створює проблеми у впровадженні нового комплексного ПЗ.

1.4. Методологія розробки систем автоматизації

Останні десятиліття, розробники та команди розробників все частіше почали стикатися з доволі поширеними сьогодні проблемами при розробці програмного забезпечення, такими як: недолік прозорості (у будь-який момент важко сказати на якому етапі знаходиться проект), недолік контролю (без чіткої оцінки процесу розробки порушуються терміни реалізації), надолік моніторингу (неможливість спостерігати за ходом розвитку проекту), неконтрольовані зміни (нові ідеї замовників інколи можуть зашкодити кінцевій реалізації проекту), недостатня надійність (найскладнішим процесом є пошук та виправлення помилок всередині коду), неправильний вибір методології розробки ПЗ (може негативно відобразитись на результатах розробки).

Існує доволі багато підходів до розробки програмного забезпечення, одним з яких є методологія гнучкої розробки програмного забезпечення. Це клас методологій що базується на ітеративній розробці із залученням до створення вимог та реалізації розв'язків цих вимог залучають багатофункціональні команди здатні до самоорганізації, і який дозволяє підвищити продуктивність співробітників і, в той же час, мінімізувавши ризики.

Основними ідеями гнучкої розробки є[19,25,26]:

- Особистісні взаємодії - більш важливі, ніж процеси та інструментарій.
- Робоче ПЗ є більш важливим, ніж повна документація продукту.
- Кооперація і співпраця із замовником є більш важливою, ніж контрактні зобов'язання.
- Швидке реагування на зміни є більш важливим, ніж дотримання плану розробки.

У 2001 році, 11-13 лютого був підписаний маніфест гнучкої розробки, який узгодилипредставники методологій Scrum, Adaptive software development, Extreme programming, DSDM, Crystal Clear, Feature driven development та Pragmatic Programming.

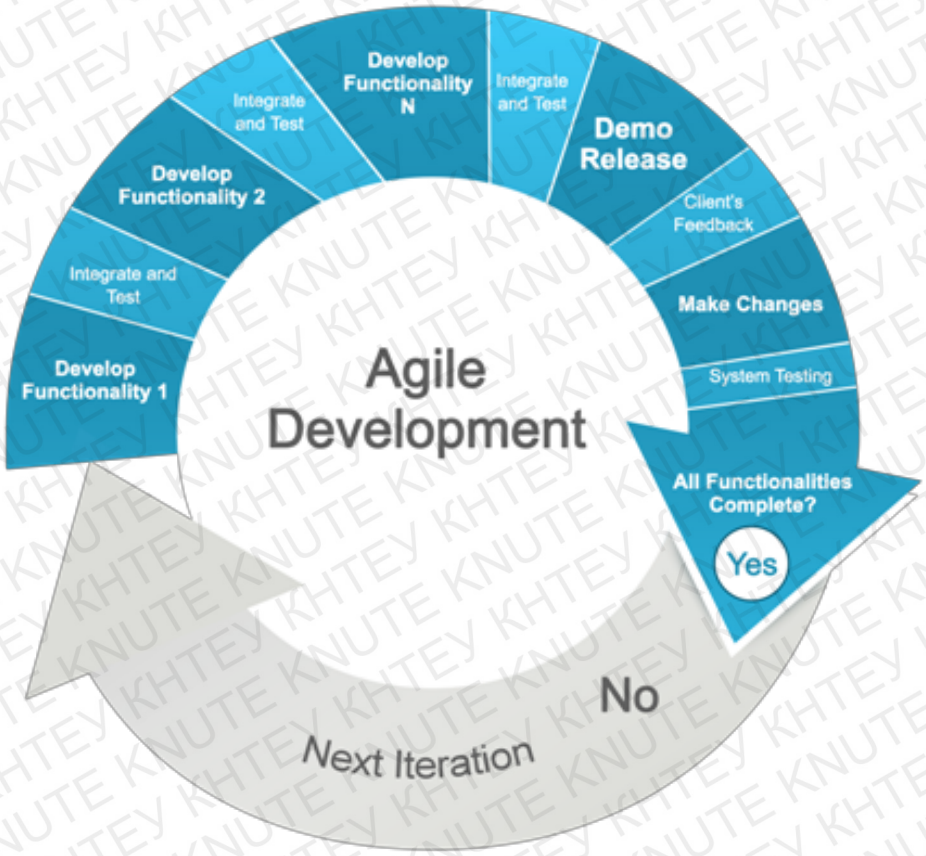

Рис 1.3 Структура AGILE

Джерело: [44]

Основні принципи, які роз'яснені у Agile Manifesto[19]:

- Задовольнити клієнта за рахунок швидкої і безперебійної поставки програмного забезпечення.
- Відкритість до змін продукту, навіть наприкінці розробки.
- Часта поставка версій ПЗ.
- Якомога частіше спілкування замовника з розробником під час розробки.
- Мотивація розробників.
- Особиста розмова - найефективніший спосіб передачі інформації.
- Найкраще мірило прогресу розробки - робоче ПЗ.
- Підвищена увага до зручного дизайну та технічної сторони проекту.

- Не робити зайвої роботи.
- Самоорганізована команда - найбільш ефективна.
- Швидка реакція на зміни.

Саме гнучкі методології привертають сьогодні увагу великих корпорацій за рахунок своєї результативності.

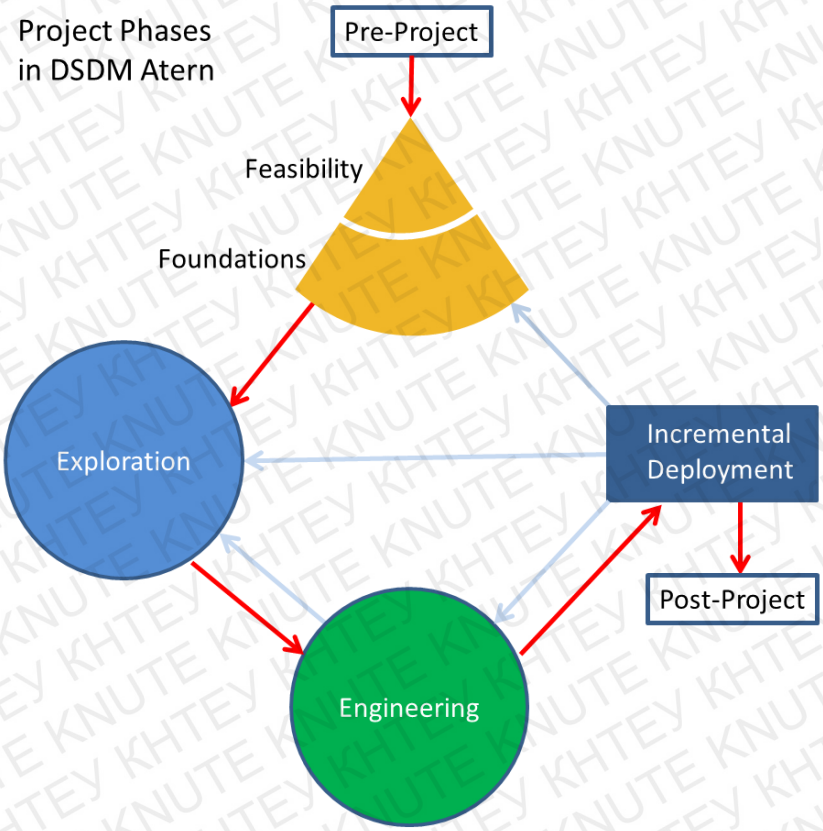

Рис 1.4 Структура DSDM

Джерело: [45]

Для створення нової ефективної методології потрібно розуміти, що автоматизована система обліку студентів - це програмне забезпечення, що може базуватися на одному або декількох підходах до створення кінцевого продукту. На даний момент, мною було обрано два підходи, комбінація яких дозволить побудувати найбільш ефективну, на мою думку, методологію розробки. Це DSDM (Dynamic Systems Development Method) та XP (Extreme programming).

DSDM - це методика розробки ПЗ, заснована на концепції швидкої розробки додатків програмних продуктів[23]. Особливостями даного методу є - побудова етапів проекту з чітко визначеними термінами реалізації та відповідним бюджетом кожного етапу, з можливістю його корегування під час розробки у відповідь на зміну вимог до продукту замовником.

Найбільше відповідають нашим вимогам такі принципи DSDM[23] як:

- Залучення кінцевого користувача
- Часта поставка версій результату
- Поставка продукту, яка задовольняє вимогам кінцевого користувача менш важлива, ніж вирішення критичних проблем у функціоналі продукту
- Розробка - ітеративна та інкрементна.
- Будь які зміни функціоналу під час розробки - можна повернути назад.
- Тестування інтегровано в життєвий цикл розробки проекту.

Для успішного використання методології потрібно налагодити взаємодію і зв’язок між розробником проекту та майбутніми користувачами і вищим керівництвом, у даному випадку адміністративним і керівним складом університету.

Extreme programming - найпопулярніша серед так званих гнучких методології, має на меті поліпшення якості ПЗ та чутливість до змін у вимогах замовників[21,22]. Як вид гнучких технологій, XP радить часті «релізи» програмного продукту у коротких циклах розробки, що поліпшує продуктивність праці та покращує можливість виконання вимог замовника, що часто змінюються.

Серед елементів екстремального програмування варто виділити ті, що нас цікавлять: проведення обширної перевірки коду та його модульне тестування, уникання створення функціональності до того, як вона стане дійсно необхідною, простота та ясність коду, часте спілкування з кінцевим користувачем.

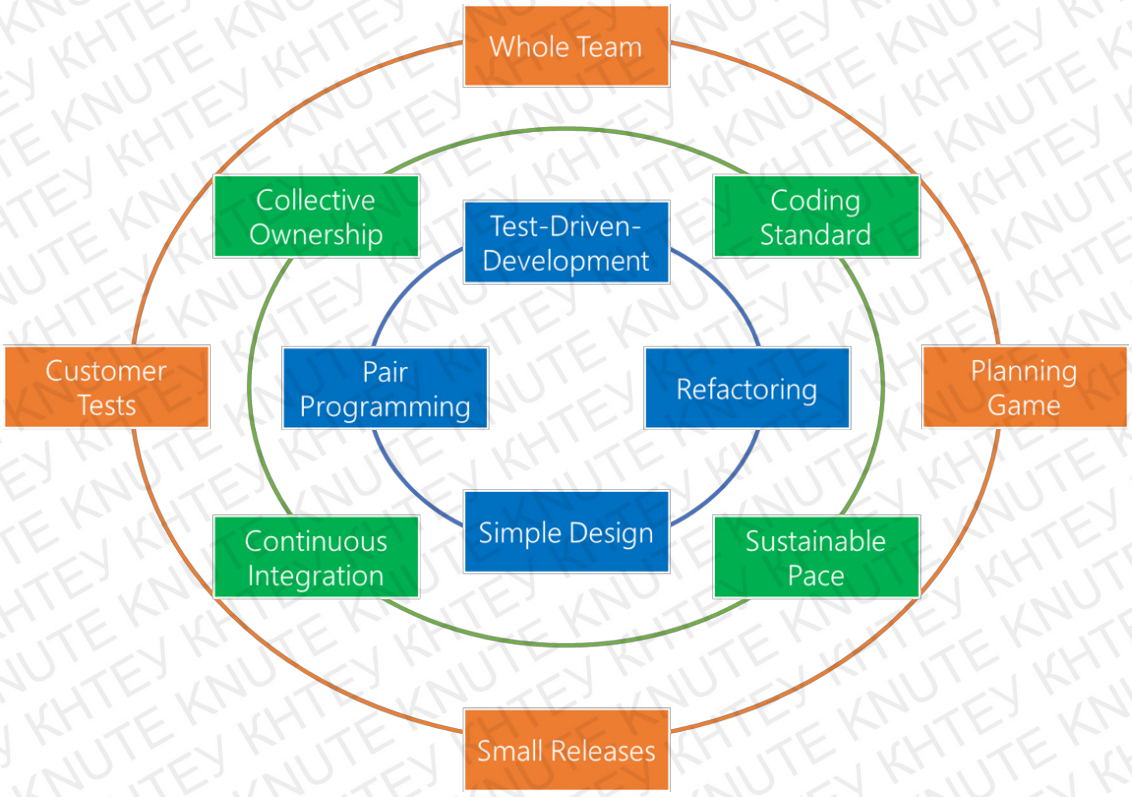

Рис 1.4 Структура XP

Джерело: [46]

Принципи XP, які найбільше відповідають нашим вимогам[21,26]:

- Розробка через тестування.
- Замовник завжди поруч.
- Безперервна інтеграція.
- Рефакторинг.
- Часті невеликі релізи.
- Простота.
- Стандарт кодування.

Комбінація цих двох методик дозволяє нівелювати деякі недоліки кожної з них, і в цілому провести ефективну розробку програмного забезпечення під вимоги кінцевого користувача, в умовах коли замовником є вищий навчальний заклад.

Основними положеннями методики розробки системи обліку є:

- Розділення життєвого циклу розробки проекту на чотири частини: дослідження, створення функціональної моделі, створення конструктивної моделі, реалізація.
- Перша частина - дослідження задачі, консультації з кінцевим споживачем, створення списку основних вимог продукту, визначення термінів розробки, створення глобальної архітектури системи і план створення прототипів. Ця стадія є найобширнішою, оскільки на даному етапі ми можемо взагалі не розуміти, що замовник хоче отримати в результаті.
- Створення функціональної моделі - визначення функціоналу прототипу, розробка прототипу та перевірка його працездатності. Результат - документ, який містить огляд прототипу, його недоліки та необхідні доопрацювання.
- Створення конструктивної моделі - розробка другої версії продукту, яка враховуватиме недоліки попередньої, можливі зміни вимог замовника внесені у ТЗ, і реалізовані у даній версії проекту. Основною особливісттю даного єтапу є тестування продукту контрольною групою кінцевих користувачів, та створення документації.
- Реалізація - виправлення недоліків виявлених на попередній стадії, затвердження протестованої системи кінцевими користувачами.
- Постійні консультації з кінцевим користувачем для внесення правок в систему і виправлення недоліків «на ходу».
- Часте постачання версій результату, використовуючи принципи безперервної інтеграції продукту.
- Ітеративна та інкрементна розробка через програмне тестування для досягнення найбільш ефективного рішення. Спочатку визначаємо вимоги, потім пишемо тести, потім реалізацію.
- Постійний рефакторинг коду і підтримка його простоти.
- Використання стандартів кодування, узгодження про найменування, принципів написання коду, певних шаблонів проектування і т.д.

- Використання систем автоматичної документації продукту

Висновки до розділу 1

Системи автоматизації (CRM - системи) дозволяють значно прискорити робочий процес, за рахунок формалізації бізнес-процесів, попередження виникнення помилок робочого процесу і побудови чіткої, логічної системи взаємодії користувача з необхідними йому інструментами роботи і структурованими робочими даними. І на сьогоднішній день подібні системи впроваджуються у всі сфери людської діяльності, від процесу навчання і до надання юридичних послуг до роздрібної торгівлі і виробництва.

Сучасний досвід використання інформаційних систем автоматизації навчального процессу є, переважно, у закордонних ВУЗ-ах, які мають потужну фінансову базу для розгортання і використання подібних проєктів (систем) у своїх процесах. В Україні найяскравішим і найвідомішим представником автоматизації процесів у сфері освіти - є ЄДЕБО, єдина державна електронна база з питань освіти, яка підтримується і фінансується Міністерством освіти і науки України.

Існує доволі багато підходів до розробки програмного забезпечення, одним з яких є методологія гнучкої розробки програмного забезпечення. Це клас методологій що базується на ітеративній розробці із залученням до створення вимог та реалізації розв’язків цих вимог залучають багатофункціональні команди здатні до самоорганізації, і який дозволяє підвищити продуктивність співробітників і, в той же час, мінімізувавши ризики.

Для створення нової ефективної методології потрібно розуміти, що автоматизована система обліку студентів - це програмне забезпечення, що може базуватися на одному або декількох підходах до створення кінцевого продукту.

РОЗДІЛ 2
АНАЛІЗ ВИМОГ ДО АВТОМАТИЗОВАНОЇ СИСТЕМИ ОБЛІКУ СТУДЕНТІВ

2.1. Аналіз та визначення задач створення проекту

Для визначення задач проекту «Автоматизована система обліку студентів» необхідно, у першу чергу, визначити цілі проекту. Цей крок дозволить визначити, який результат ми отримаємо після завершення проекту, і які дії нам необхідно виконати для досягнення цієї цілі. Цілі і задачі мають бути чіткими заявами про наміри. Кожна ціль має мати власний напрям, який впливає на кінцевий результат усього проекту. Цілі і задачі мають бути вимірюваними.

Кожна ціль має відповідати на питання «Що?». Іншими словами, що буде робити проект?

В свою чергу задачі представляють собою конкретні дії, які приводять до виконання цілі. Кожна задача може мати одну, або декілька пов’язаних з нею задач. По суті, задача це визначення «як» буде виконуватись проект.

Цілі розробки автоматизованої системи обліку студентів:

- Збереження інформації про студентів на електронних носіях - це дозволить уникнути зайвої паперової роботи з картотекою студентів і пришвидшити пошук необхідної інформації[7,1]:
    - Проаналізувати тип інформації що потрібно зберегти[30] - провівши попередній аналіз інформації що необхідно зберегти, ми одразу будемо розуміти як і де нам її зберігати. Для прикладу строкові типи даних зберігаються у реляційних базах даних, а документи і динамічні формати - у не-реляційних базах даних. В нашому випадку ми будемо зберігати відомості про студентів і викладачів, які можна зберігати у текстовому та цифровому вигляді, а також дати.

- Проаналізувати об’єкти які будуть сформовані - всю інформацію яку ми вирішимо зберігати нам необхідно привести до об’єктної структури. А саме створити об’єкти даних якими ми будемо оперувати на рівні додатку (web-інтерфейсу) і на відомості про яких будемо опиратися при конфігуруванні бази даних[31,9].
- Проаналізувати кількість і типи параметрів - необхідно визначити кількість та тип параметрів кожного об’єкту програми, а також зрозуміти яку дію вони будуть виконувати.
- Конфігурувати базу даних під потреби проекту - проаналізувавши наші потреби, ми зрозуміємо як і яким чином конфігурувати нашу базу даних, яку архітектуру побудови таблиць нам потрібно використати для реалізації цілі[9].
- Створити інтерфейс доступу до бази даних - на програмному рівні необхідно створити інтерфейс доступу до бази даних, а також реалізувати поєднання таблиць у базі даних з об’єктами і класами програми. Вирішити яке архітектурне рішення ми будемо використовувати для з’єднання. Ми будемо поєднувати класи програми з таблицями, чи з відображеннями? В нашому проекті ми будемо використовувати змішаний підхід[35].

- Швидкий і зручний доступ до інформації - використання web-інтерфейсу простить і пришвидшить доступ до необхідної оператору інформації за рахунок автоматизованих форм:
  - Визначити тип необхідної інформації - необхідно зрозуміти яка інформація потрібна користувачу, а яка залишиться суто системною. І відповідно тим чи іншим чином налаштувати контролери доступу додатку.
  - Створити web-інтерфейс - необхідно створити простий і зручний спосіб вносити інформацію до бази даних, без використання спеціальних програм, а саме web-інтерфейс, у якому за допомогою форм користувач буде мати змогу це зробити.

- Можливість редагувати інформацію - використання web-інтерфейсу з чітко визначеними формами дозволяє просто і без помилок вносити зміни до інформації:
    - Проаналізувати тип інформації яка буде змінюватись - окрім внесення нових даних, користувачу також необхідно змінювати данні. Для реалізації такої можливості необхідно визначити, які саме данні планують змінювати, а які будуть залишатись статичними протягом значних проміжків часу, і які можна змінювати за допомогою спеціальних програм.
    - Реалізувати web-інтерфейс для редагування інформації - створити простий і зручний web-інтерфейс для внесення змін в динамічні данні в базі, не використовуючи спеціальні застосунки.

Послідовне виконання завдань проекту дозволить якісно та швидко виконати необхідні 80 відсотків роботи над проектом, і створити функціональний прототип, який можна віддати на тестування кінцевому користувачу.

Основною функцією проекту є автоматизація ручного процесу зберігання даних студентів, і удосконалення існуючих підходів до виконання монотонної роботи методистами університету. Адже автоматизація процесу дозволяє зекономити фізичні ресурси людини, за рахунок виконання частини роботи оператора, підвищення комфорту роботи з даними і мінімізації введення механічних помилок в систему.

2.2. Розробка технічного завдання для системи автоматизації

Технічне завдання (ТЗ) - вихідний документ для проектування споруди чи промислового комплексу, конструювання технічного пристрою (приладу, машини, системи керування тощо), розробки автоматизованої системи, створення програмного продукту або проведення науково-дослідних робіт (НДР) у відповідності до якого проводиться виготовлення, приймання при введенні в дію та експлуатація відповідного об'єкту або проекту.

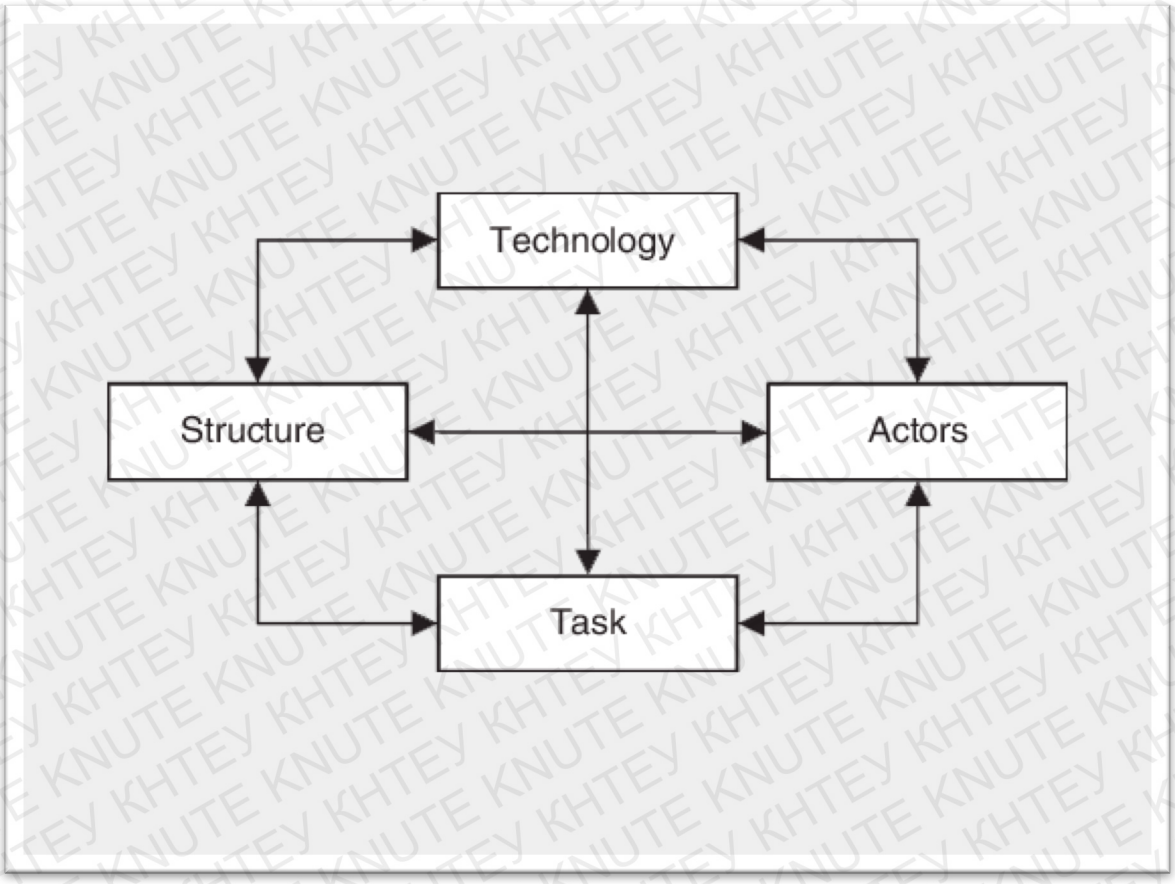

Рис 2.1 Взаємодія складових ТЗ

Джерело: [розроблено автором]

Згідно з діючим стандартами ТЗ повинно містити у собі наступні відомості про об'єкт розробки[7,40]:

- Найменування об'єкту розробки, та область застосування:
  - повне найменування об'єкту та його умовне позначення;
  - шифр теми або шифр (номер) договору;
  - перелік документів, на підставі яких створюється проект, ким і коли затверджені ці документи;
  - планові терміни початку та закінчення робіт із створення об'єкту.
- Підстава для розробки та назва проектної організації:
  - найменування підприємств розробника і замовника системи та їхні реквізити;

- перелік юридичних та фінансових документів, на підставі яких створюється система, ким і коли затверджені ці документи;
- відомості про джерела та порядок фінансування робіт.

- Мета розробки.
- Джерела розробки. Тут повинні бути перераховані документи та інформаційні матеріали (техніко-економічне обґрунтування, звіти про закінчені науково-дослідні роботи, інформаційні посилання на вітчизняні й зарубіжні аналоги та інше), на підставі яких розроблялося ТЗ і які мають бути використані при створенні системи.
- Технічні вимоги, які включають:
  - склад об'єкту та вимоги до його конструктивного виконання;
  - показники призначення та економічного використання сировини, матеріалів, палива і енергії;
  - вимоги до надійності;
  - вимоги до технологічності;
  - вимоги до рівня уніфікації і стандартизації;
  - вимоги безпеки при роботі обладнання;
  - естетичні й ергономічні вимоги;
  - вимоги до складових частин продукції, сировини і експлуатаційних матеріалів;
  - вимоги патентної чистоти;
  - вимоги експлуатації, вимоги до технічного обслуговування і ремонту;
  - вимоги до категорії якості.
- Економічні показники:
  - гранична ціна;
  - економічний ефект;
  - термін окупності витрат на розробку і освоєння об'єкту;
  - допустима річна потреба в об'єкті проектування.
- Порядок контролю і приймання об'єкту:

- види, склад, обсяг і методи випробувань системи та її складових частин (види випробувань відповідно до діючих норм, які поширюються на систему, що розробляється);
- загальні вимоги до приймання робіт (продукції) за стадіями (перелік учасників, місце і терміни проведення), порядок узгодження і затвердження приймальної документації;
- статус приймальної комісії.

Технічне завдання - це вихідний документ для розробки нового програмного забезпечення (у даному випадку), в якому формуються основні цілі розробки, список принципових вимог до продукту, визначаються терміни та етапи розробки і регламентується процес приймально-здавальних випробувань. У складанні технічного завдання беруть участь як представники замовника, так і представники виконавця. Цей документ містить основні вимоги замовника, вихідні дані для розробки, в ТЗ вказуються призначення продукту, область його застосування, стадії розробки різної документації, її склад, терміни виконання тощо, а також особливі вимоги, зумовлені специфікою проекту або умовами його експлуатації. Як правило ТЗ складається на основі аналізу передових досягнень техніки, результатів виконання попередніх досліджень, науково-дослідних робіт, наукового прогнозування тощо.

Технічне завдання дозволяє (як сполучна ланка між замовником і виконавцем):

Обом сторонам:

- представити готовий проект до початку роботи;
- виконати попунктно перевірку готового продукту
- зменшити кількість помилок, пов'язаних зі зміною вимог внаслідок їхньої неповноти або хибності.

Замовнику:

- усвідомити, що саме йому потрібно, чітко це сформулювати;
- вимагати від виконавця відповідності продукту всім обумовленим та затвердженим пунктам ТЗ.

Виконавцю:

- зрозуміти суть поставленого завдання;
- планувати виконання проекту в деталях і працювати за наміченим планом;
- відмовитися від виконання робіт, не зазначених у ТЗ.

Технічне завдання на розробку програм складається, перш за все, для тих людей, які будуть здійснювати цю саму розробку. Відповідно, воно повинно бути зрозумілим тій людині, яка нічого не знає про клієнта, і про його завдання і проблеми[36].

Отже, технічне завдання на розробку програми має розповісти виконавцю і про фірму, і про цілі, і про завдання. При цьому чим конкретніше буде розповідь, тим краще - і для оповідача, чи Замовника розробки програм, і для слухача, тобто для Виконавця проекту.

У загальному вигляді, технічне завдання переслідує декілька цілей:

- організація;
- інформація;
- комунікація;
- юрисдикція.

Організацію повинно бути спрямовано на сам процес, інакше кажучи, упорядкувати творчість і творення розроблюваної програми або програмного комплексу. Структура технічного завдання на розробку програм повинна бути чіткою і в той же час лаконічною.

Інформаційна складова ТЗ повинна бути повною, але стислою[36]. І знову ж таки просте правило, "необхідно і достатньо". Його, як водиться, потрібно дотримуватися завжди і скрізь, але при складанні технічного завдання з розробки програми, це правило стає номер один. Грамотне технічне завдання - перший і останній документ, який розповість про всі бажання замовника у зручній для розуміння програміста формі.

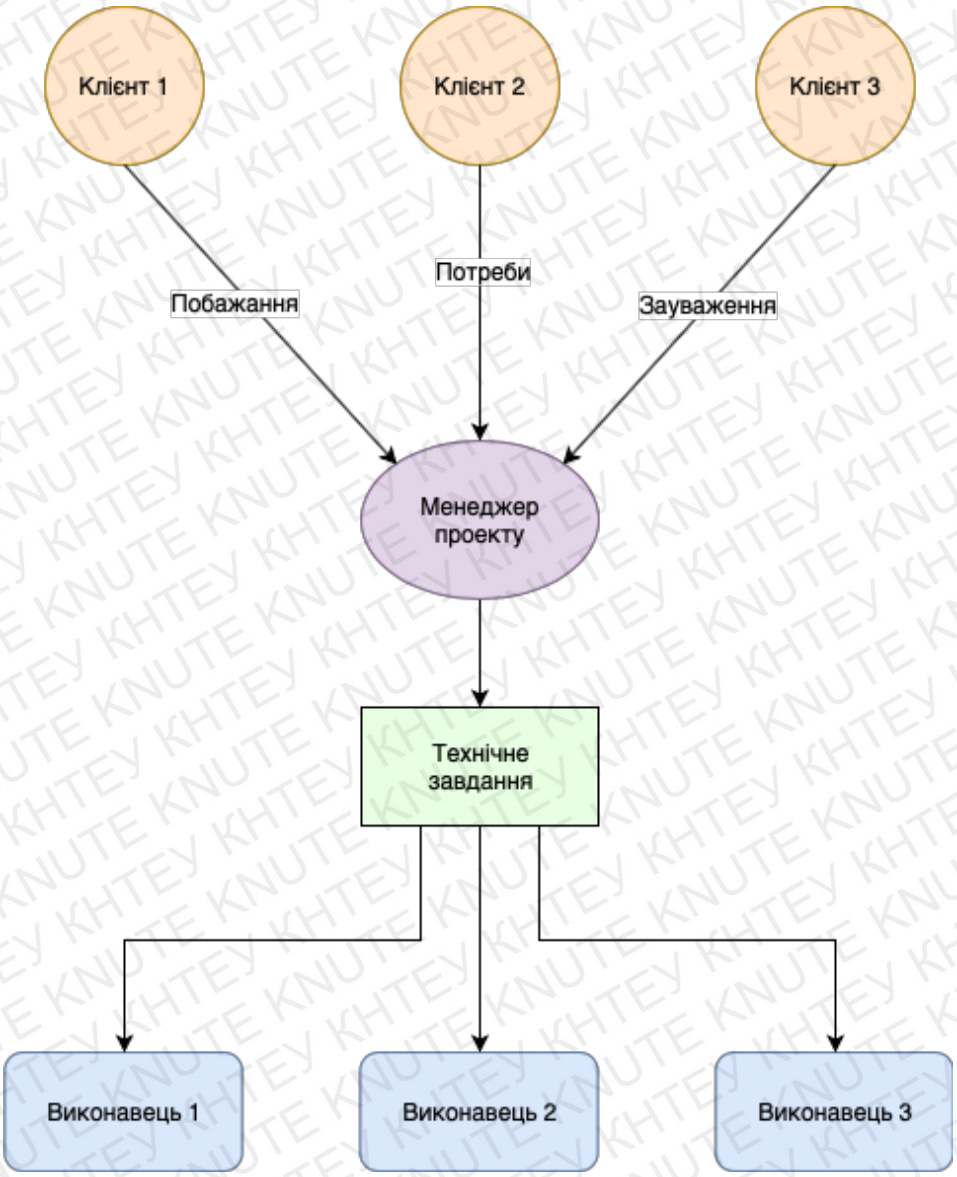

Рис 2.2 Схема створення і використання ТЗ

Джерело: [розроблено автором]

З комунікаціями дещо складніше, тому що комунікації завжди складні. Особливо, якщо спілкуватися різними мовами. А мов тут може бути декілька, більш точно - за кількістю учасників проекту під кодовою назвою "розробка програми". Простіше кажучи: Клієнт, він же Замовник, Менеджер проекту, Виконавці проекту: програміст(и), інші можливі учасники, які мають рацію, як зробити, як зробити краще, і чим все має закінчитися[38].

Природно, створюючи спільний проект, ці учасники змушені шукати мову,

доступну для загального розуміння кожним. Такою мовою і покликане стати технічне завдання на розробку програми. Головне - встановити канал зв'язку між першою і третьою ланкою, і чим менше перешкод при цьому буде вносити друга і четверта ланки, тим якіснішим буде результат, а розробка програми принесе бажаний результат при мінімальних витратах ресурсів.

Завдяки технічним завданням, можна судити про відповідність результату розробки програм і заданих початкових умов. Загальна структура технічного завдання, у тому числі й на розробку програм, містить в собі:

- що потрібно зробити в рамках проекту - потрібно створити автоматизовану систему обліку студентів;
- навіщо це потрібно, і для яких конкретно цілей - розробка проекту проводиться з ціллю автоматизації організаційного процесу в межах університету;
- де буде використовуватися результат проекту (читай, розробка програм), в якій сфері діяльності, і на якому рівні - реалізоване програмне забезпечення планується використовувати в сфері надання освітніх послуг, на адміністративному рівні;
- які вимоги повинна задовольняти розробка програм - простота у використанні, можливість подальшого масштабування і розробки;
- що потрібно зробити в процесі роботи над проектом - конфігурувати сервер бази даних, конфігурувати базу даних, реалізувати програмні контролери взаємодії з базою даних з програмного рівня, розробити додаток і контролери взаємодії з візуальною частиною web-додатку;
- як буде оцінюватися результат з боку замовника - результат буде оцінюватись кінцевим користувачем;
- якими документами встановлюється порядок взаємодії - специфікацією проекту;
- на чому заснована ініціація роботи над проектом по розробці програм - змішана методологія розробки програмного забезпечення DSDM та XP.

Окремим пунктом можна виділити розділ вимог до програмного забезпечення.

При складанні цього розділу, до питання необхідно підходити формально.

Також розробка програм повинна задовольняти ряду вимог, які необхідно викласти в технічному завданні. До приблизного списку вимог можна віднести:

- набір виконуваних програмою функцій - повний облік студентів і збереження наданої інформації у базі даних, облік викладачів, предметів та навчальних груп, зручний і швидкий доступ до інформації, реалізація клієнт-серверного додатку, створення web-клієнту;
- організація вхідних та вихідних даних - текстова, цифрова та часова інформація, створення програмних класів для їх тимчасового зберігання, постійне збереження їх у базі даних;
- швидкодія - реалізація такої конфігурації клієнт-сервер при якої взаємодія між додатком та базою даних буде максимально швидким;
- надійність функціонування - реалізація такої конфігурації клієнт-сервер при якої функціонування додатку та бази даних буде максимально надійним (можливе використання дублюючих серверів, тощо.),
- тривалість відновлення при відмовах - використання дублюючих серверів для функціонування бази даних та додатку;
- відмови у зв'язку з некоректними діями користувача - відсутність будь-якої роботи додатку у зв’язку з некоректними діями користувача;
- види обслуговування - серверне адміністрування, адміністрування бази даних, адміністрування системи;
- число і кваліфікація персоналу, що взаємодіє з програмою - мінімум один технічний спеціаліст та велика кількість користувачів (1-1000), знання технічним спеціалістом Unix системи Linux, субд MySQL, мов програмування Java, JavaScript, знання web-технологій;
- параметри технічних засобів, на яких буде забезпечуватися нормальна працездатність програми:
  - Два віддалених сервери під керуванням системи Linux Ubuntu, для

створення дублюючої системи баз-данних:

- 8 gb оперативної пам’яті
- 4 CPUs
- 100+ gb HDD disk;

- Два віддалених сервери під керуванням системи Linux Ubuntu, для створення дублюючої системи web-додатків:
  - 2 gb оперативної пам’яті
  - 1 CPUs
  - 10+ gb HDD disk;

- вихідні мови та коди програмування, інформаційні структури і сторонні програмні засоби:
  - Java
  - JSP
  - JSTL
  - Maven
  - MySQL
  - JavaScript
  - JQuery
  - HTML
  - CSS
  - NGinx
  - Twitter Bootstrap
  - Intellij Idea
  - Data Grip
  - MySQL Workbench
  - SSH Shell
  - Insomnia;
- захист та інформаційна безпека - використання правильно налаштованих

серверів, непрямий доступ у базу даних;

- умови транспортування і зберігання - можливість швидкої міграції і запуску.

Рис 2.3 Варіації можливих серверів

Джерело: [47]

Також список вимог на розроблення програм може бути змінений: доповнений або скорочений залежно від конкретних умов проекту.

До технічного завдання можуть входити і такі розділи:

- Допущення і обмеження. Як правило, цей розділ заповнюється Виконавцем, однак, Замовнику теж важливо знати про призначення цього розділу. Будь-яка розробка програмного забезпечення ведеться в деяких обмеженнях. Це дозволяє не збільшувати вартість до нескінченності, а також довести проект до логічного кінця. У

цьому розділі перераховують як правило наступні припущення та обмеження: перераховується функціональність, що виходить за рамки проекту, завдання, що виходять за рамки проекту, технічні обмеження, залежно від зовнішніх умов, які можуть вплинути на прийняті зобов'язання.

- Ризики. Це фактори, які можуть вплинути на вартість і терміни виконання робіт. Ризики часто описуються у наступному форматі: заголовок, ідентифікація ризику, ймовірність ризику, вартість, порядок дій при виникненні ризику. Як приклад ризику можна навести інтеграцію системи з програмним забезпеченням сторонніх розробників.

2.3. Розробка моделей бізнес-процесів

Під бізнес процесом розуміється набір логічно взаємопов’язаних дій або завдань, виконання яких призводить до очікуваного результату. Тому, практично всі процеси організації можна віднести до бізнес процесів.

Основна мета управління бізнес процесами полягає у приведенні процесів у відповідність з цілями організації. Кожен процес повинен бути налаштований таким чином, щоб результати процесу приводили до досягнення бізнес цілей.

Залежно від ознаки класифікації бізнес процеси розділяються за видами. Управління бізнес процесами поширюється на всі з них.

Процеси управління. Ці процеси призначені для планування, моніторингу та аналізу роботи. За рахунок процесів управління можна гарантувати досягнення цілей виробничими і процесами забезпечення[25]. Процеси управління не додають цінності для кінцевого споживача, але вони необхідні для результативної та ефективної роботи організації. До таких процесів часто відносять процеси планування, постановки цілей, моніторингу та вимірювань, бюджетування та ін.

Виробничі (основні) процеси. За рахунок цих бізнес процесів організація досягає своїх цілей. Виробничі процеси забезпечують перетворення продукту або послуги і

додають цінності для кінцевого споживача[25]. До виробничих процесів відносять процеси проектування, виготовлення, надання послуг, монтажу та ін.

Процеси забезпечення. Ці процеси необхідні для нормального виконання виробничих процесів. Вони не приносять доданої цінності для кінцевого споживача, однак без них неможливе досягнення цілей виробничих процесів[25]. До забезпечує процесів відносять процеси закупівлі, управління персоналом, управління інфраструктурою та ін.

Управління бізнес процесами складається з наступних фаз[39,14]:

- Перша фаза – визначення процесу. На цій фазі виконується моделювання процесу в початковому стані і в бажаному стані (розробляються моделі «як є» і «як має бути»);
- Друга фаза – аналіз процесу. На цій фазі визначаються різні варіанти дій процесу, проводиться імітаційне моделювання. У результаті визначаються оптимальні методи для поліпшення бізнес процесу.
- Третя фаза – реалізація змін. На цій фазі до процесу застосовуються вибрані методи поліпшення. Відбувається впровадження змін у процес.
- Четверта фаза – моніторинг процесу. На цій фазі виконується періодичний моніторинг процесу за певними показниками.
- П’ята фаза – оптимізація процесу. На цій фазі виконується порівняння реально отриманих результатів по зміні процесу з бажаною моделлю («як повинно бути») і починається наступний цикл поліпшення.

Метою моделювання бізнес-процесів як правило є[41,27]:

- Документація бізнесу компанії
  - Для отримання знання про бізнес
  - Формування карти підрозділів
  - Переведення бізнесу в інші місця
  - Для задоволення потреб бізнес-партнерів або об'єднань (наприклад, з метою сертифікації)

- Для навчання співробітників (передачі знань)
- Для впровадження (підтримки системи менеджменту якості) та екологічного менеджменту

- Підготовка бізнес-процесів (який зазвичай починається з аналізу фактичного стану)
  - З метою впровадження нових організаційних структур
  - Впровадження аутсорсингу
- Підготовка і автоматизації ІТ-підтримки бізнес систем
- Визначення показників процесу
- Бенчмаркінг
- Найкраща практика
- Організаційні зміни
  - При підготовці до продажу бізнесу
  - При підготовці до інтеграції компаній або їх частин
  - Введення або зміна ІТ-систем та/або організаційних структур
- Участь у конкурсах (наприклад, Європейський фонд управління якістю)
- Удосконалення внутрішніх процесів.

Будь-яку компанію (бізнес) можна представити як якийсь чорний ящик, що вміщає в себе сукупність бізнес-процесів, де на виході – прибуток. На питання «Що на вході?», «Що всередині?», «Як вона працює?» допомагає відповісти опис бізнес-процесів.

Моделювання та опис бізнес-процесів[27,25] – це, перш за все, інформаційна база для аналітика, але не мета проекту. Щоб розробка моделі бізнес-процесів була виправдана, а сама модель згодом ефективно застосовується, необхідно чітко сформулювати її мету, точку зору, межі предметної області та глибину деталізації.

Модель бізнес-процесів і опис бізнес-процесів, розроблені компанією BSC, дають відповіді на наступні питання:

- Які процедури (функції, роботи) необхідно виконати для отримання заданого

кінцевого результату;

- В якій послідовності виконуються ці процедури;
- Які механізми контролю та управління існують в рамках описуваного бізнес-процесу;
- Які вхідні документи / інформацію використовує кожна процедура бізнес-процесу;
- Які вихідні документи / інформацію генерує процедура бізнес-процесу;
- Які ресурси необхідні для виконання кожної процедури бізнес-процесу;
- Яка документація / умови регламентує виконання процедури;
- Які параметри характеризують виконання процедур і бізнес-процесу в цілому.

Для побудови моделей бізнес-процесів і опису бізнес-процесів компанія BSC використовує методології SADT, сімейства IDEF, DFD, UML, ARIS та інші.

UML (Unified Modeling Language) – це об’єктно-орієнтована графічна мова для візуалізації, специфіціювання, конструювання та документування систем, де велика роль відводиться опису бізнес-процесів в інформаційних системах[8]. UML є мовою широкого профілю, це відкритий стандарт, який використовує графічні позначення для створення абстрактної моделі системи, яка називається UML моделлю. UML був створений для визначення, візуалізації, проектування та документування здебільшого програмних систем[16].

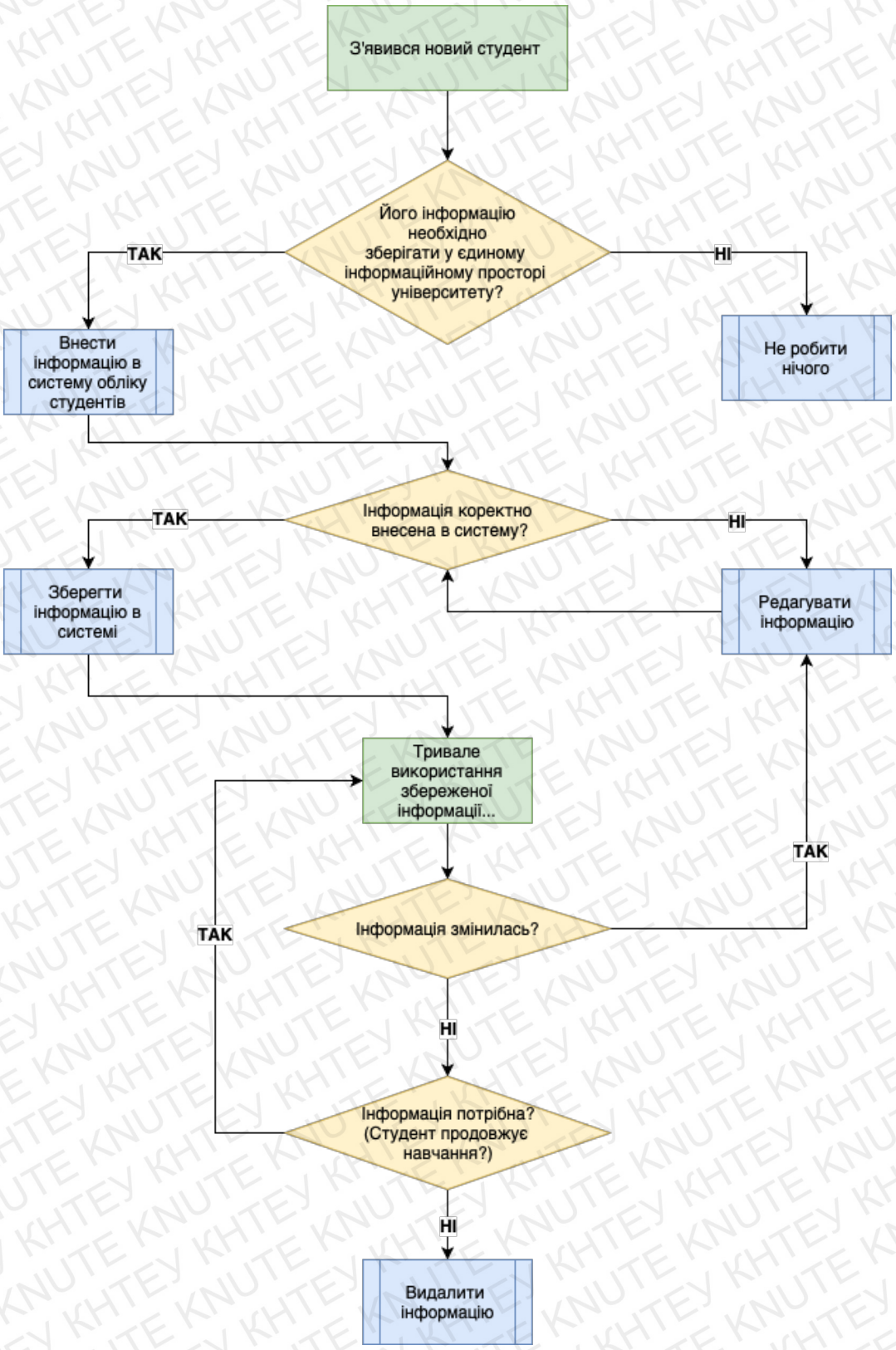

Схема 2.4 Бізнес-процес використання системи обліку студентів

Джерело: [розроблено автором]

Склад методики:

- моделювання предметної області;
- вимоги до системи;
- аналіз та проектування;
- тестування;
- запуск.

Моделювання бізнес-процесів дозволяє проаналізувати не тільки, як працює підприємство в цілому, як воно взаємодіє із зовнішніми організаціями, замовниками та постачальниками, але і як організована діяльність на кожному окремо взятому робочому місці.

Моделювання бізнес-процесів організації включає два етапи: структурне і детальне.

Під методологією (нотацією) створення моделі (опису) бізнес-процесу розуміється сукупність способів, за допомогою яких об'єкти реального світу і зв'язку між ними представляються у вигляді моделі.

Основу багатьох сучасних методологій моделювання бізнес-процесів склала методологія SADT (Structured Analysis and Design Technique - метод структурного аналізу і проектування) та алгоритмічні мови, застосовувані для розробки програмного забезпечення. За допомогою методології сімейства IDEF можна ефективно відображати і аналізувати моделі діяльності широкого спектру складних систем в різних розрізах. Система ARIS являє собою комплекс засобів аналізу і моделювання діяльності підприємства. Її методичну основу становить сукупність різних методів моделювання, що відображають різні погляди на досліджувану систему[16].

Необхідно враховувати важливі характеристики моделювання бізнес-процесів. Зокрема, до переваг моделювання бізнес-процесів відносять:

- підвищення якості та швидкості виробництва продукції з одночасним

зниженням витрат,

- зростання професіоналізму співробітників;
- підвищення конкурентоспроможності компанії.

Недоліки, у свою чергу:

- посилення експлуатації співробітників і пов'язані з цим проблеми соціально-психологічного характеру;
- необхідність проведення цілеспрямованої роботи по зміні корпоративної культури.

2.4. Розробка діаграми діяльності

Діаграма діяльності це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів[14]. Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності[13]. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності. Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграми активності будуються з обмеженої кількості фігур, з’єднаних стрілочками.

Найважливіші типи фігур:

- округлені прямокутники позначають дії;
- ромби позначають рішення
- риски позначають початок (розподіл) чи кінець (об’єднання) паралельних активностей;
- чорний кружок позначає старт (початковий стан) процесу;
- чорний кружок в колі позначає кінець (кінцевий стан).

Стрілки ведуть від старту до кінця і позначають порядок в якому відбуваються активності.

Діаграма активності може вважатись формою блок-схеми[14].

Мова уніфікованого моделювання містить кілька підмножин діаграм, включаючи структурні діаграми, діаграми взаємодії та діаграми поведінки. Діаграми активності разом з діаграмами використання та державними машинами розглядаються як діаграми поведінки, оскільки вони описують, що повинно відбутися в моделюванні системи.

Діаграми активності представляють ряд переваг для користувачів[13,16]:

- Демонструють логіку алгоритму.
- Описують кроки, які потрібно виконати у випадку використання UML.
- Ілюструють бізнес-процес або робочий процес між користувачами та системою.
- Спрощують та вдосконалюють будь-який процесс шляхом уточнення складних випадків використання.
- Можуть візуально зображувати моделі елементів архітектури програмного забезпечення, такі як метод, функція та операція.

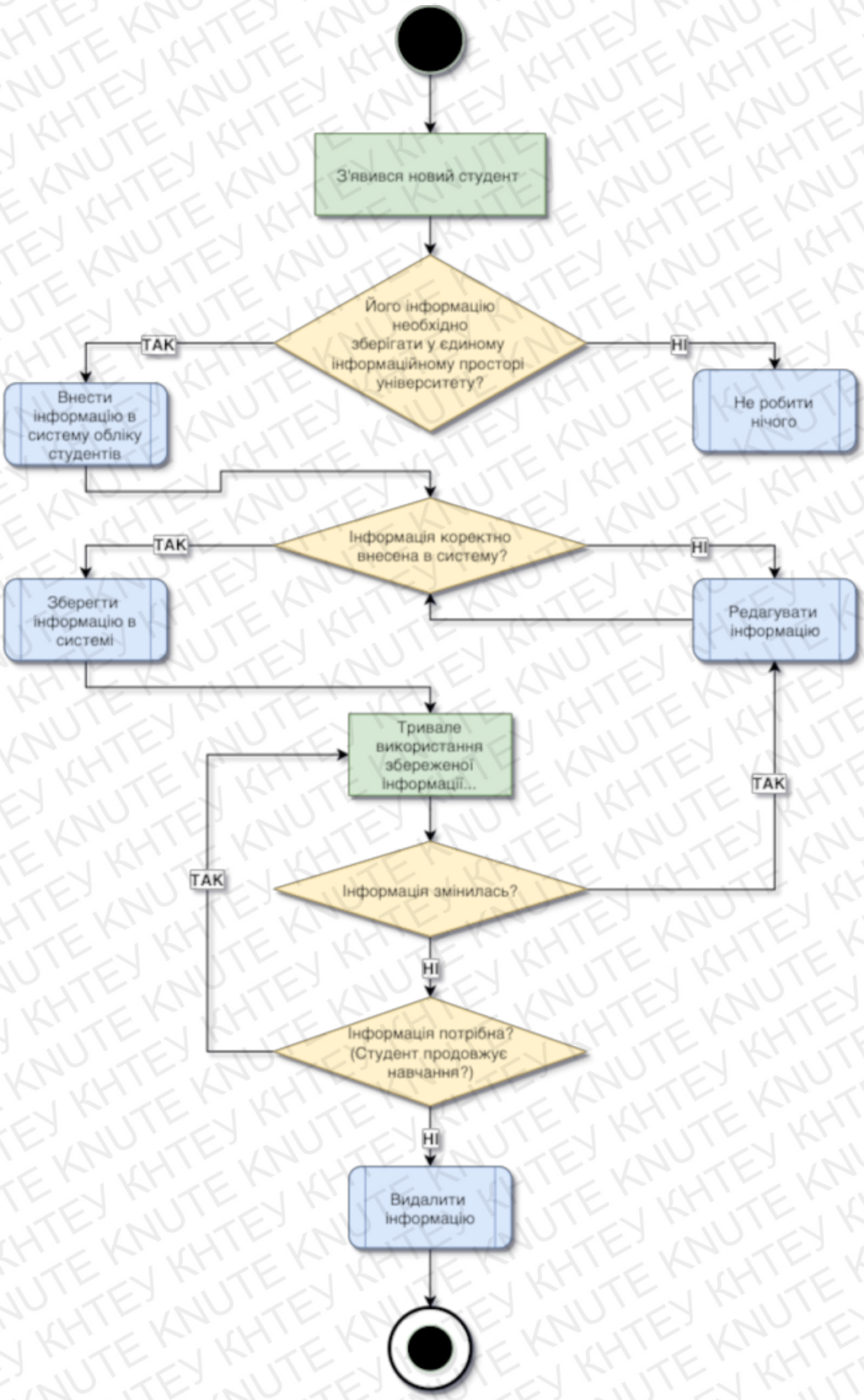

Схема 2.5 Діаграма діяльності системи обліку студентів

Джерело: [розроблено автором]

Висновки до розділу 2

Для успішної реалізації проекту «Автоматизована система обліку студентів» необхідно, у першу чергу, визначити цілі та задачі проекту. Цей крок дозволить визначити, який результат ми отримаємо після завершення проекту, і які дії нам необхідно виконати для досягнення цієї цілі. Цілі і задачі мають бути правильно поставлені і бути чіткими заявами про наміри. Кожна ціль має мати власний напрям, який впливає на кінцевий результат усього проекту. Цілі і задачі мають бути вимірюваними.

Послідовне виконання завдань проекту дозволить якісно та швидко виконати необхідні 80 відсотків роботи над проектом, і створити функціональний прототип, який можна віддати на тестування кінцевому користувачу.

Одним з найголовніших задач – є побудова технічного завдання, вихідного документу для розробки нового програмного забезпечення (у даному випадку), в якому формуються основні цілі розробки, список принципових вимог до продукту, визначаються терміни та етапи розробки і регламентується процес приймально-здавальних випробувань. Цей документ містить основні вимоги замовника, вихідні дані для розробки, в ТЗ вказуються призначення продукту, область його застосування, стадії розробки різної документації, її склад, терміни виконання тощо, а також особливі вимоги, зумовлені специфікою проекту або умовами його експлуатації.

Також важливою частиною побудови будь-якого проекту – є розуміння бізнес процесів, які необхідно реалізувати і автоматизувати у програмному забезпеченні[27].

Під бізнес процесом розуміється набір логічно взаємопов’язаних дій або завдань, виконання яких призводить до очікуваного результату. Тому, практично всі процеси організації можна віднести до бізнес процесів.

Основна мета управління бізнес процесами полягає у приведенні процесів у відповідність з цілями організації. Кожен процес повинен бути налаштований таким чином, щоб результати процесу приводили до досягнення бізнес цілей.

Процеси управління. Ці процеси призначені для планування, моніторингу та аналізу роботи. За рахунок процесів управління можна гарантувати досягнення цілей виробничими і процесами забезпечення[27]. Процеси управління не додають цінності для кінцевого споживача, але вони необхідні для результативної та ефективної роботи організації. До таких процесів часто відносять процеси планування, постановки цілей, моніторингу та вимірювань, бюджетування та ін.

Виробничі (основні) процеси. За рахунок цих бізнес процесів організація досягає своїх цілей. Виробничі процеси забезпечують перетворення продукту або послуги і додають цінності для кінцевого споживача. До виробничих процесів відносять процеси проектування, виготовлення, надання послуг, монтажу та ін.

Моделювання бізнес-процесів дозволяє проаналізувати не тільки, як працює підприємство в цілому, як воно взаємодіє із зовнішніми організаціями, замовниками та постачальниками, але і як організована діяльність на кожному окремо взятому робочому місці[5].

Ще одним важливим моментом, який дозволяє краще підготуватись до реалізації проекту є діаграма діяльності, яка є візуальним представленням графу діяльності. Граф діяльності є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів. Подібна специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Архітектура проекту і створення макету проекту

Архітектура програмного забезпечення (англ. software architecture) - це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін стосується також документування архітектури програмного забезпечення. Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами, дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах[1].

Основною архітектури програмування є ідея зниження складності системи шляхом абстракції і розмежування повноважень.

В кінцевому підсумку архітектура створюється для задоволення комплексу потреб зацікавленої особи. Однак часто неможливо виконати всі виражені побажання. Наприклад, зацікавлена особа може попросити, щоб деяка функціональність укладалася в певний часовий проміжок, але ці дві потреби (функціональність і проміжок часу) є взаємовиключними. Можна або зменшити кордону функції, щоб вона відповідала розкладу, або надати повну функціональність, але за більш тривалий відрізок часу. Аналогічно, різні зацікавлені особи можуть мати суперечливі потреби і тут має бути досягнуто певної рівноваги. Тому прийняття компромісних рішень є необхідним аспектом процесу розробки архітектури.

Зазвичай усі зацікавлені особи мають такі вимоги і побажання до кінцевої архітектури ПЗ:

- Кінцевий користувач зацікавлений в інтуїтивно зрозумілому і коректному поведінці, продуктивності, надійності, зручності використання, доступності та

безпеці;

- Системний адміністратор зацікавлений в інтуїтивно зрозумілому поведінці, управлінні та інструментах моніторингу;
- Спеціаліст з маркетингу зацікавлений в конкурентноспроможних функціях, часу до виходу програми, позиціонуванні серед інших продуктів і у вартості;
- Клієнт зацікавлений в ціні, стабільності і можливості планувати;
- Розробник зацікавлений у зрозумілих вимогах і простому і несуперечливою принципі проектування;
- Керівник проекту зацікавлений в передбачуваності ходу проектування, плануванні, продуктивному використанні ресурсів і бюджету;
- Спеціаліст з супроводу зацікавлений в зрозумілому, несуперечливою і документованої принципі проекту, а також в легкості, з якою можна вносити зміни.

Як видно зі списку, ще одна проблема розробника – це те, що зацікавлені особи зацікавлені не тільки в тому, щоб система забезпечувала необхідну функціональність. Багато пункти зі списку інтересів є нефункціональними за характером, так як вони не впливають на функціональність системи (наприклад, інтерес по відношенню до ціни та планування). Тим не менш, такі інтереси формулюють властивості або обмеження системи. Нефункціональні вимоги дуже часто є найбільш значущими вимогами, оскільки в них зацікавлений розробник архітектури.

Важливий аспект архітектури – це не тільки кінцевий результат, тобто сама архітектура, а й її логічне обґрунтування. Таким чином, важливо забезпечити документування рішень, які привели до створення цієї архітектури, і логічні обґрунтування таких рішень[28,41].

Більшість архітектур побудовані на основі систем, які використовують подібні набори інтересів. Подібність може бути визначене як архітектурний стиль, який можна розглядати як особливий вид шаблону, хоча цей шаблон часто є складним і складовим (коли одночасно застосовуються декілька шаблонів).

Система розміщується в деякому оточенні, і це оточення впливає на архітектуру. Іноді це називають “архітектурою в контексті”. В основному, оточення визначає межі, в яких повинна працювати система, а це, в свою чергу, впливає на архітектуру. Фактори оточення, мають вплив на архітектуру – це місія бізнесу, яку буде підтримувати архітектура, зацікавлені в системі особи, внутрішні технічні обмеження (наприклад, вимога відповідати стандартам організації) і зовнішні технічні обмеження (такі як необхідність взаємодіяти із зовнішньою системою або відповідати зовнішнім регулятивним нормам)[39].

В області системного проектування компроміс досягається за рахунок використання програмного забезпечення, апаратного забезпечення та людей. Наприклад, якщо ключовим фактором є продуктивність, то приймається рішення реалізувати певні елементи системи у вигляді апаратних пристроїв, а не програм або людей. Ще один приклад: щоб надати зручну для покупців систему, приймається рішення надати клієнтський інтерфейс у вигляді людини, а не програми або обладнання[37,36]. Більш складні сценарії вимагають домогтися певних характеристик системи через поєднання програмного, апаратного забезпечення і людини. (Відповідно до цього, в даному циклі статей в деяких випадках даються посилання на елементи, які не є програмним забезпеченням.)

Слід також зазначити, що кожна система має архітектуру, навіть якщо ця архітектура формально не документована або система занадто проста, і, скажімо, складається з одного елемента. Зазвичай документування архітектури являє собою дуже цінний засіб. Документовані архітектури мають тенденцію бути більш продуманими – а, отже, більш ефективними – чим недокументовані, оскільки процес записи архітектури природним чином веде до всебічного обмірковування[15].

І навпаки, якщо архітектура не документується, важко (якщо не неможливо) довести, що вона відповідає затвердженим вимогам в термінах адресних характеристик, таких як зручність обслуговування, запозичення передового досвіду і так далі. Архітектури, які не документувалися, як, здається, більшість з тих, що

існують на сьогоднішній день, мають тенденцію бути випадковими, а не продуманими.

Проект «Автоматизована система обліку студентів» має конкретну клієнт-серверну архітектуру зображену на діаграммі.

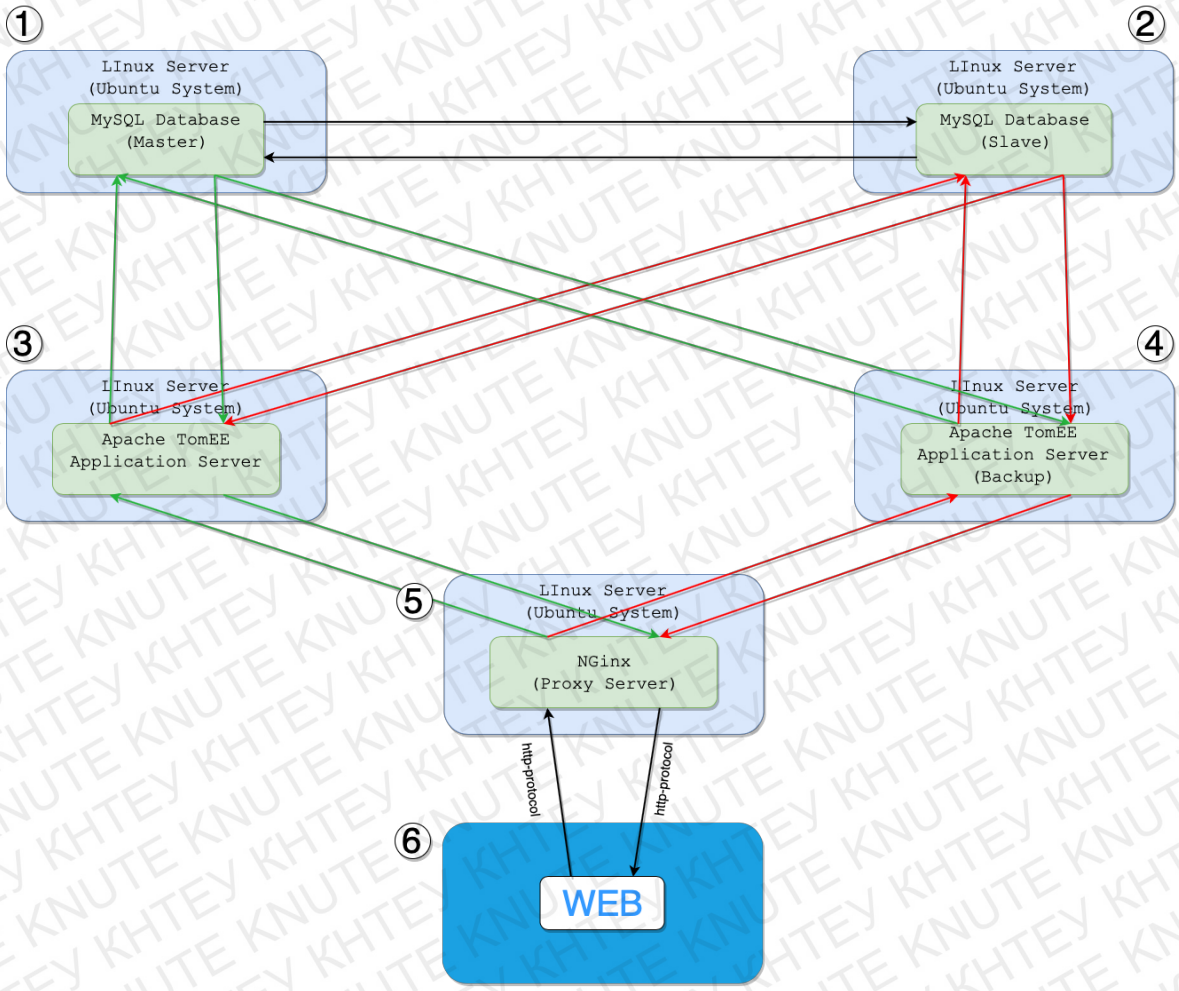

Схема 3.1 Схема взамодії серверів проекту

Джерело: [розроблено автором]

- Об'єкт №1 на діаграмі – це основний сервер бази даних (master), який відповідає за зберігання даних.

- Об'єкт №2 на діаграмі – це дублюючий сервер бази даних (slave), який відповідає за дублювання інформації яка знаходиться на основному сервері (master). Дублювання інформації має відбуватись кожні 10-15 хвилин, що має забезпечити безперебійність роботи бази даних, і забезпечити максимально можливе збереження

даних.

- Об’єкт № 3 на діаграмі – це основний сервер додатку, який відповідає за реалізацію користувацького інтерфейсу, і взаємодію з базою даних.
- Стрілки між об’єктами 1 і 3 – це основний канал взаємодії серверу додатку з базою даних, у випадку відмови роботи основного серверу (master) або його недоступності, сервер додатку автоматично має перемикати канали взаємодії з БД на дублюючий сервер (об’єкт №2)
- Об’єкт №4 – дублюючий сервер додатку, схема його взаємодії з базами даних так ж як і основного серверу додатку.
- Об’єкт №5 – проксі сервер (Nginx), який є вхідною точкою в систему. Він відповідає за аналіз доступності серверів додатку (на діаграмі об’єкти №3 і №4). За нормальних умов потік даних направляється на основний сервер TomEE у випадку відсутності, або недоступності першого серверу - потік даних перенаправляється на дублюючий Apache TomEE сервер.
- Об’єкт №6 – глобальна мережа.

Макет проекту у даній реалізації складається з:

- Конфігурації бази даних
- Конфігурації об’єктів серверу додатку

Його можна відобразити за допомогою діаграм конфігурації БД та діаграм об’єктів додатку.

Фактично конфігурація бази даних складається з декількох модулів:

- Модуль авторизації
- Модуль студентів
- Модуль викладачів
- Модуль адресного простору
- Додаткові (допоміжні) модулі

Як можна побачити на діаграмах – основним є модуль студентів, так як він

відповідає за збереження основної інформації, інформації про студентів. Менш значущим є модуль викладачів, оскільки він є додатковим модулем системи, який спрощує збереження інформації про викладачів, їх предмети, кількість годин, і т.д. Модуль адресного простору відповідає за коректне заповнення адрес проживання студентів. На даний момент в системі є інформація про 9,5 мільйонів актуальних адрес України.

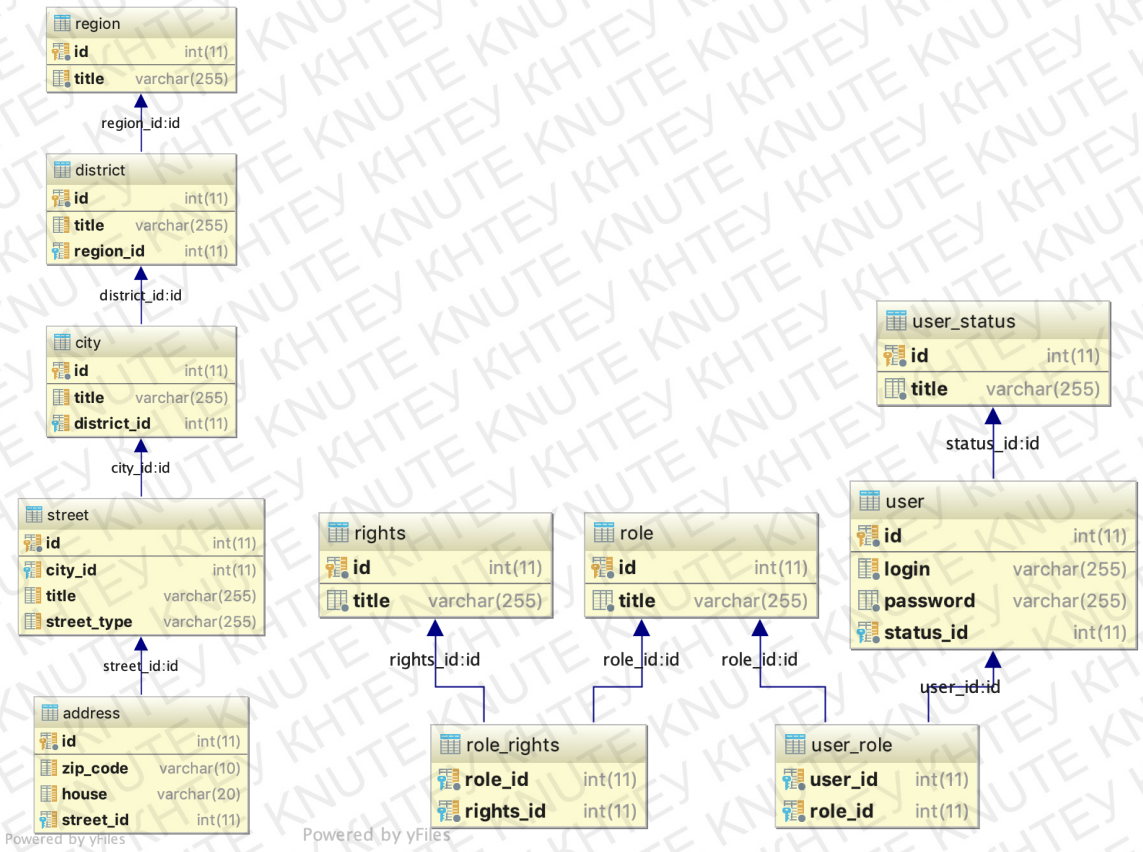

Рис 3.2 Діаграми ролей, користувачів та адрес

Джерело: [розроблено автором]

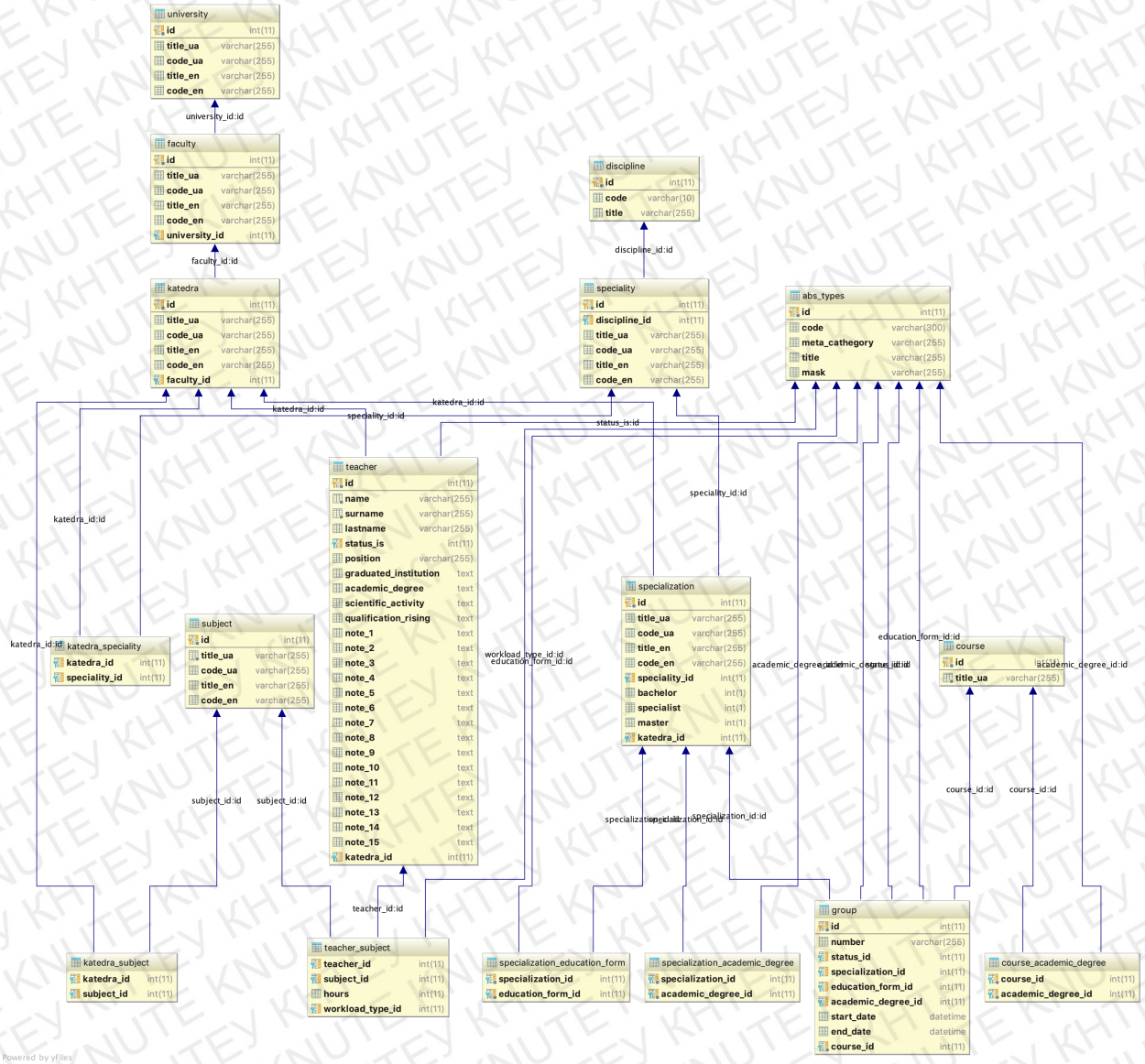

Рис 3.3 Діаграми викладачів, груп, факультетів, кафедр і т.д.

Джерело: [розроблено автором]

Модуль авторизації є необхідним, для контролю входу в систему, і забезпечення входу в неї, її використання, внесення і зміну даних в ній.

В свою чергу діаграми об’єктів серверу додатку мають дещо інший вигляд.

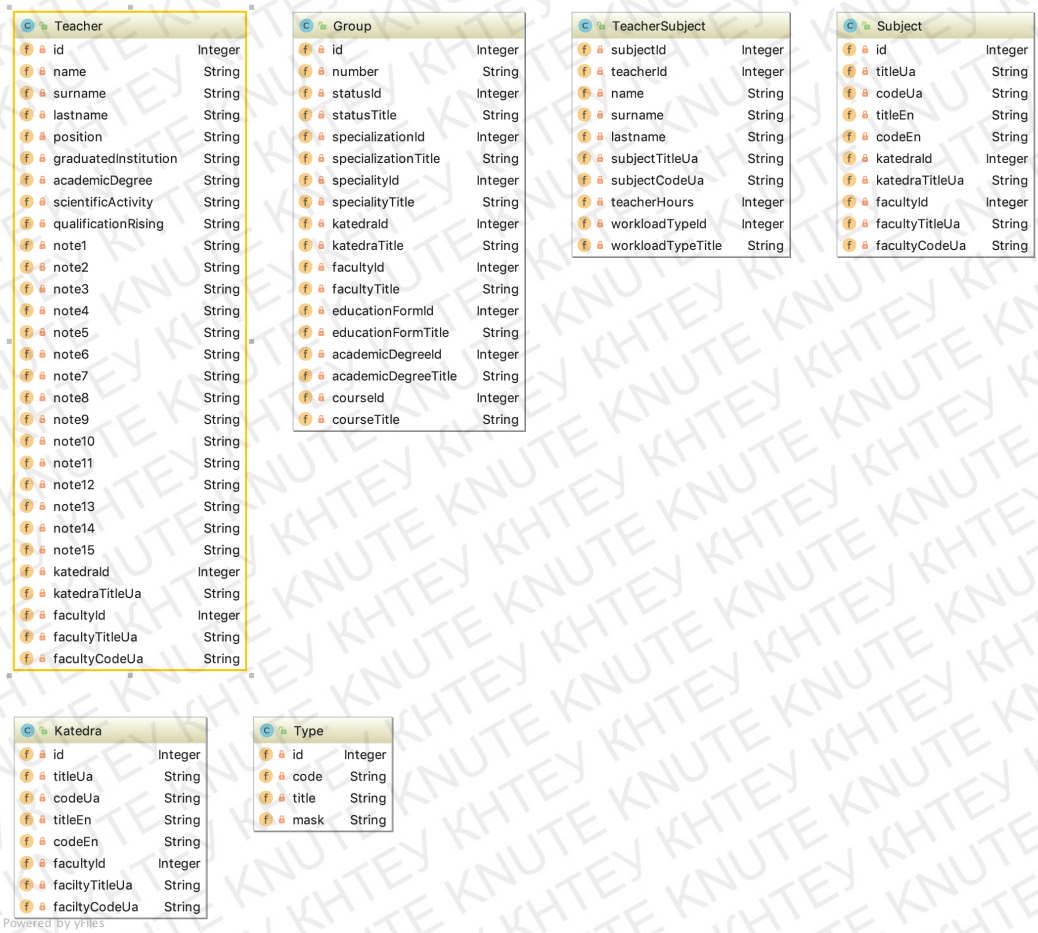

Рис 3.4 Програмні класи викладачів, груп, предметів і т.д.

Джерело: [розроблено автором]

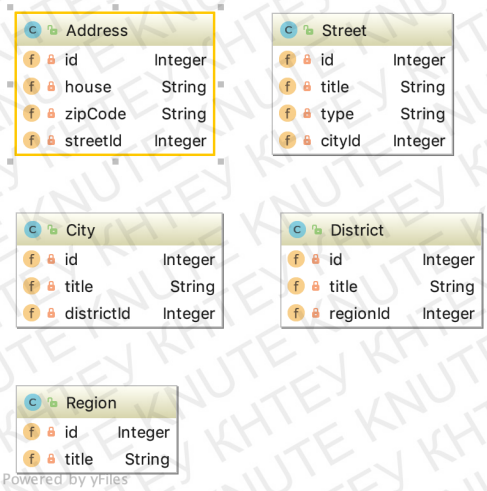

Рис 3.5 Програмні класи модулю адресного простору.

Джерело: [розроблено автором]

Діаграми контролерів взаємодії серверу додатку і бази даних також мають трохи інший формат, і відображають не просто класи об’єктів а класи контролерів та інтерфейсів доступу до БД, основних методів, і типів даних, що використовуються.

Рис 3.6 Програмні класи-контроллери

Джерело: [розроблено автором]

3.2. Реалізація програмного продукту

У процесі реалізації продукту будуть використовуватись такі технології як:

- Java - об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License. Під «незалежністю від архітектури» мається на увазі те, що програма, написана на мові Java, працюватиме на будь-якій підтримуваній апаратній чи системній платформі без змін у початковому коді та перекомпіляції. Стандартні бібліотеки забезпечують загальний спосіб доступу до таких платформозалежних особливостей, як обробка графіки, багатопотоковість та роботу з мережами[30,31,32,34,33].
- JSP - технологія, що дозволяє веб-розробникам динамічно генерувати HTML, XML та інші веб-сторінки. Робота над JSP розпочалась в 1997 році. Згодом JSP було включено у склад Java EE — програмної платформи для програмування веб-додатків[1]. Технологія дозволяє вставляти Java-код, в статичний вміст сторінки. Також можуть використовуватись бібліотеки JSP тегів для вставки їх в JSP-сторінки. Сторінки компілюються JSP-компілятором в сервлети, які є Java-класами, і виконуються на сервері. Сервлети також можуть бути написані розробником, не використовуючи JSP-сторінки. Ці технології доповнюють одна одну.
- JSTL - розширення специфікації JSP, що додає бібліотеку JSP тегів для загальних потреб, таких як розбір XML даних, умовна обробка, створення циклів і підтримка інтернаціоналізації.
- Maven - засіб автоматизації роботи з програмними проектами, який спочатку використовувався для Java проектів. Використовується для управління (management) та складання (build) програм. Створений 2002 року Джейсоном ван Зилом. За

принципами роботи кардинально відрізняється від Apache Ant, та має простіший вигляд щодо build-налаштувань, яке надається в форматі XML. XML-файл описує проект, його зв'язки з зовнішніми модулями і компонентами, порядок будування (build), папки та необхідні плагіни. Сервер із додатковими модулями та додатковими бібліотеками розміщується на серверах. Раніше Maven, де він був частиною Jakarta Project.

- MySQL - вільна реляційна система управління базами даних. Розробка та підтримка сайта MySQL здійснює корпорація Oracle, яка отримала права на торговельну марку разом з поглиненої Sun Microsystems, яка раніше придбала шведську компанію MySQL AB. Продукт поширюється як під GNU General Public License, так і під власною комерційною ліцензією[9].

- JavaScript - динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

- JQuery - популярна JavaScript-бібліотека з відкритим сирцевим кодом, яка надає можливості для розробникам створювати плагіни у верхній частині бібліотеки JavaScript. Використовуючи ці об'єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня, що сприяє створенню потужних і динамічних веб-сторінок.

- HTML - стандартизована мова розмітки документів у Всесвітній павутині.

Більшість веб-сторінок містять опис розмітки на мові HTML (або XHTML). Мова HTML інтерпретується браузерами; отриманий в результаті інтерпретації форматований текст відображається на екрані монітора комп'ютера або мобільного пристрою.

- CSS - використовується творцями веб-сторінок для завдання кольорів, шрифтів, розташування окремих блоків і інших аспектів представлення зовнішнього вигляду цих веб-сторінок. Крім того, CSS дозволяє представляти один і той же документ в різних стилях або методах виведення, таких як екранне уявлення, друковане подання, читання голосом (спеціальним голосовим браузером або програмою читання з екрану), або при виведенні пристроями, що використовують шрифт Брайля.
- NGinx - вільний веб-сервер і проксі-сервер.
- Twitter Bootstrap - це безкоштовний набір інструментів з відкритим кодом, призначений для створення веб-сайтів та веб-додатків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-додатків.
- Intellij Idea - комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala) від компанії JetBrains.
- Data Grip - IDE-інструмент для роботи з базами даних MySQL, PostgreSQL, Oracle, SQL Server, Sybase, DB2, SQLite, HyperSQL, Apache Derby і H2[9].
- MySQL Workbench - інструмент для візуального проектування баз даних, що інтегрує проектування, моделювання, створення й експлуатацію БД в єдине безкоштовне оточення для системи баз даних MySQL[9].

Після створення функціонального прототипу проекту ми отримали такі функціональні сторінки і модулі системи як:

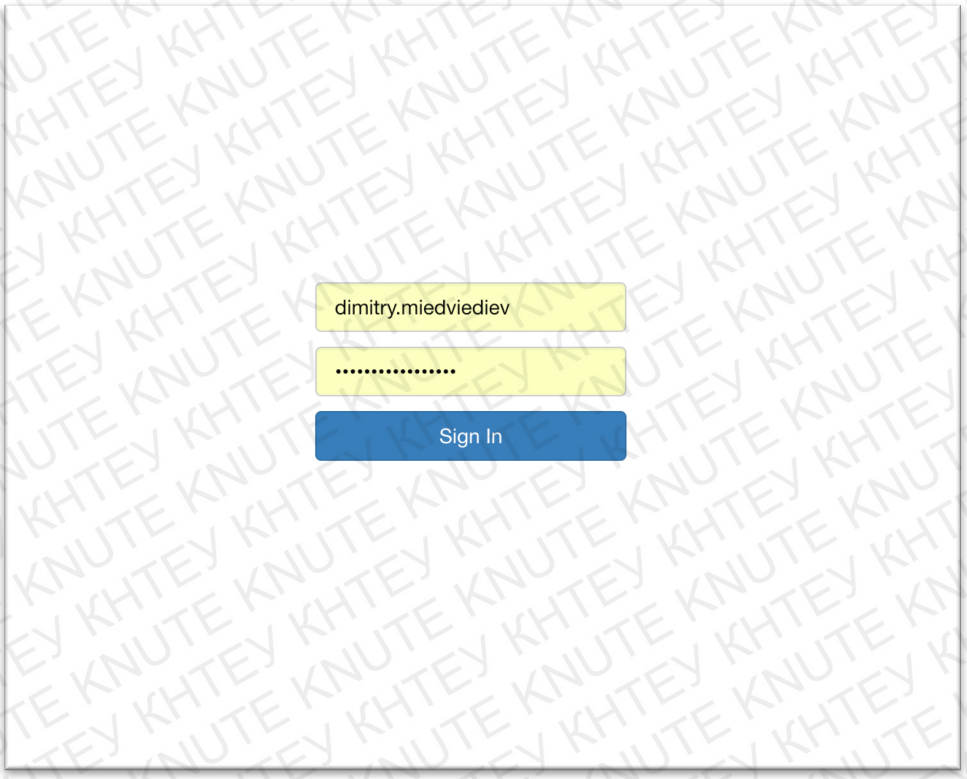

Рис 3.7 Сторінка авторизації

Джерело: [розроблено автором]

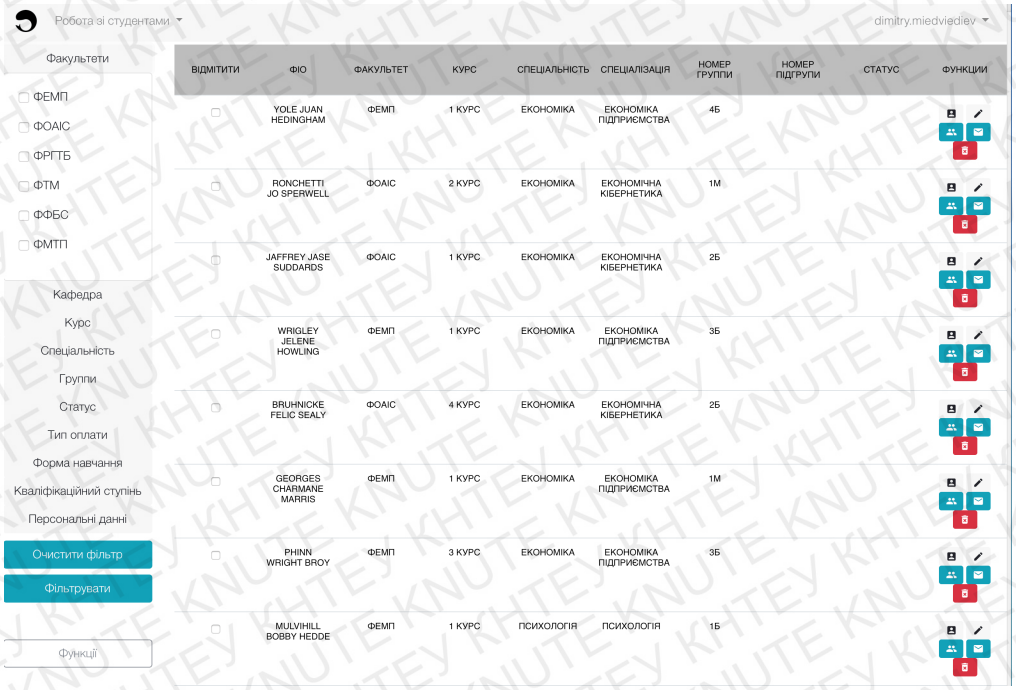

Рис 3.8 Модуль роботи зі студентами

Джерело: [розроблено автором]

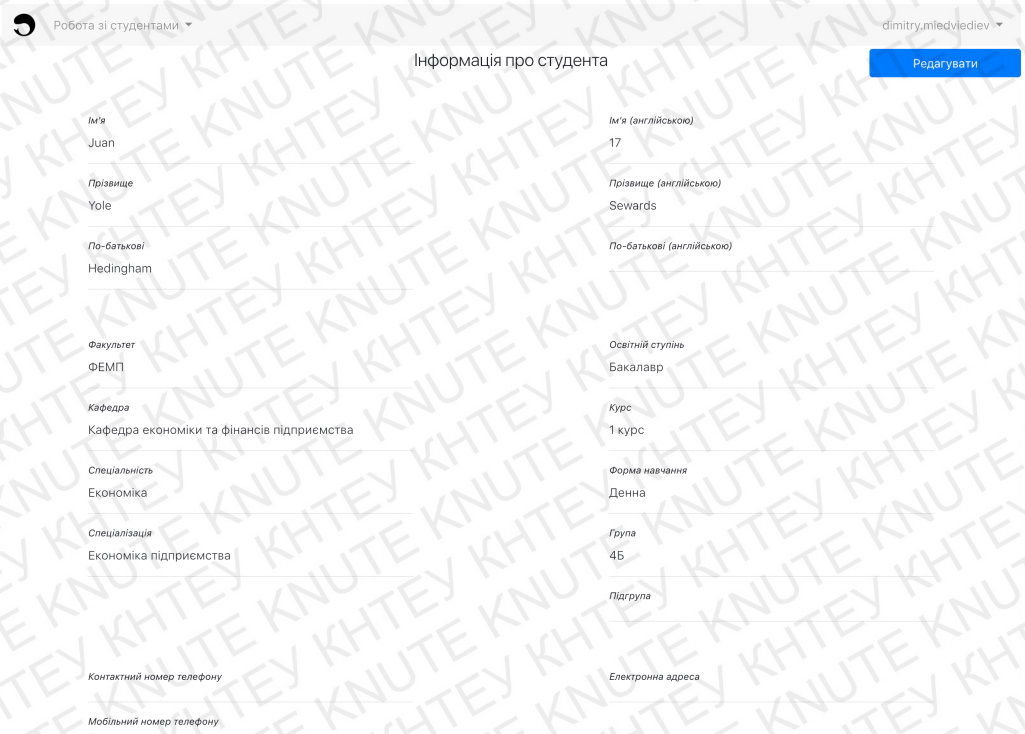

Рис 3.9 Профіль студента

Джерело: [розроблено автором]

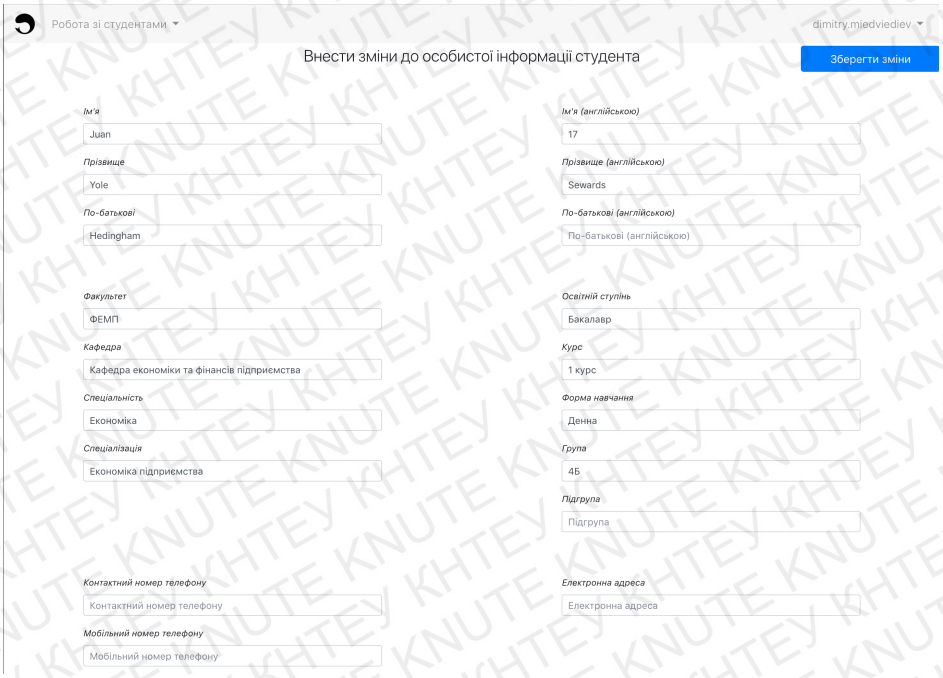

Рис 3.10 Сторінка редагування профілю студента

Джерело: [розроблено автором]

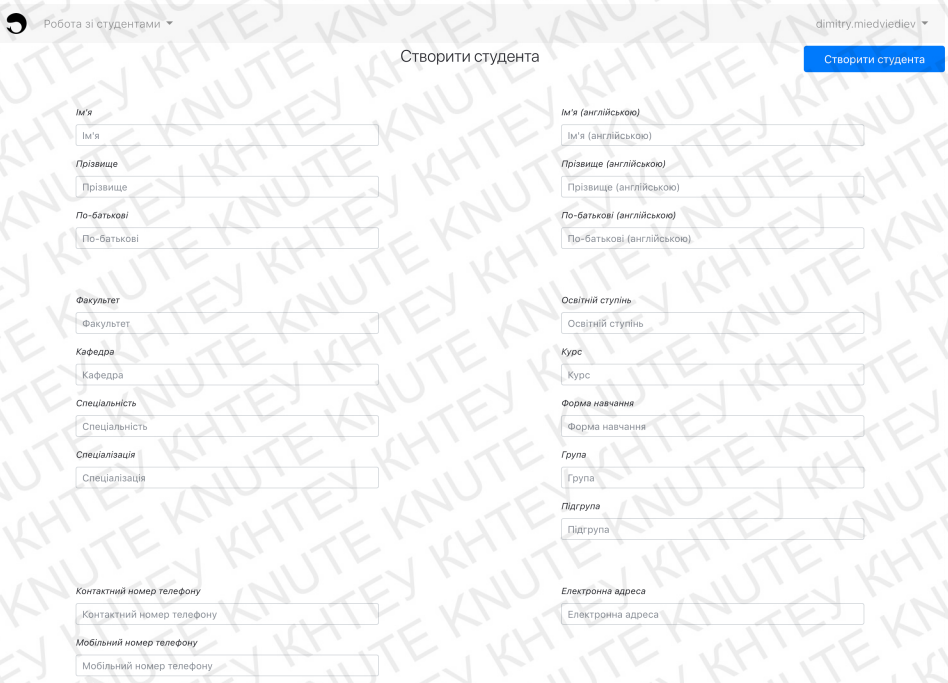

Рис 3.11 Сторінка створення студента

Джерело: [розроблено автором]

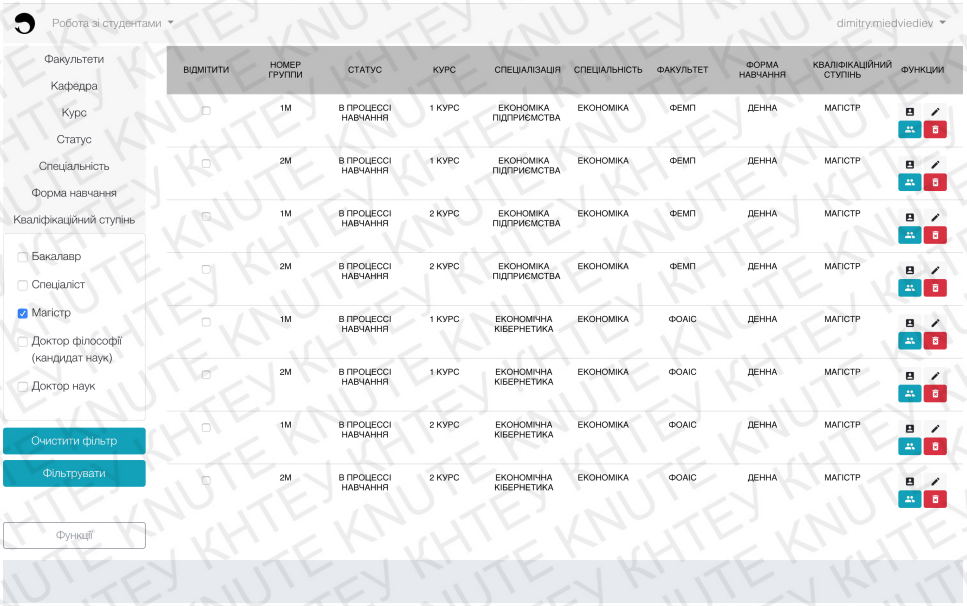

Рис 3.12 Модуль роботи із списком груп

Джерело: [розроблено автором]

Рис 3.13 Профіль груп

Джерело: [розроблено автором]

Рис 3.14 Сторінка редагування профілю груп

Джерело: [розроблено автором]

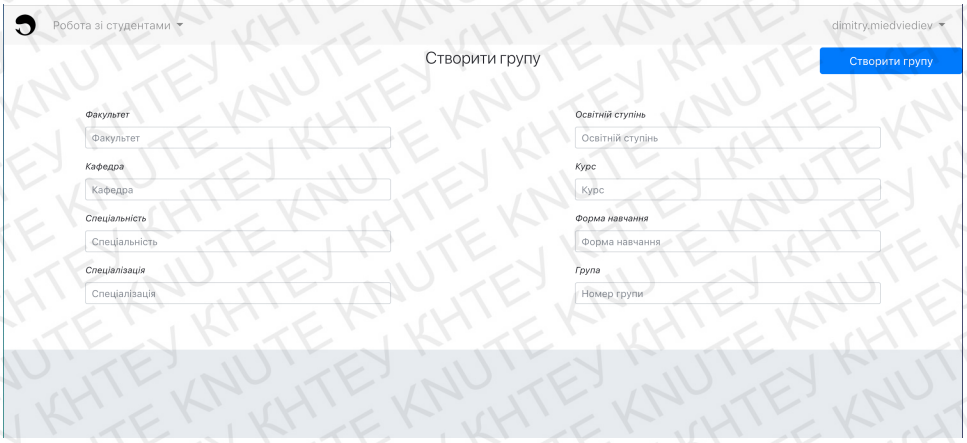

Рис 3.15 Сторінка створення групи

Джерело: [розроблено автором]

Рис 3.16 Модуль роботи із списком викладачів

Джерело: [розроблено автором]

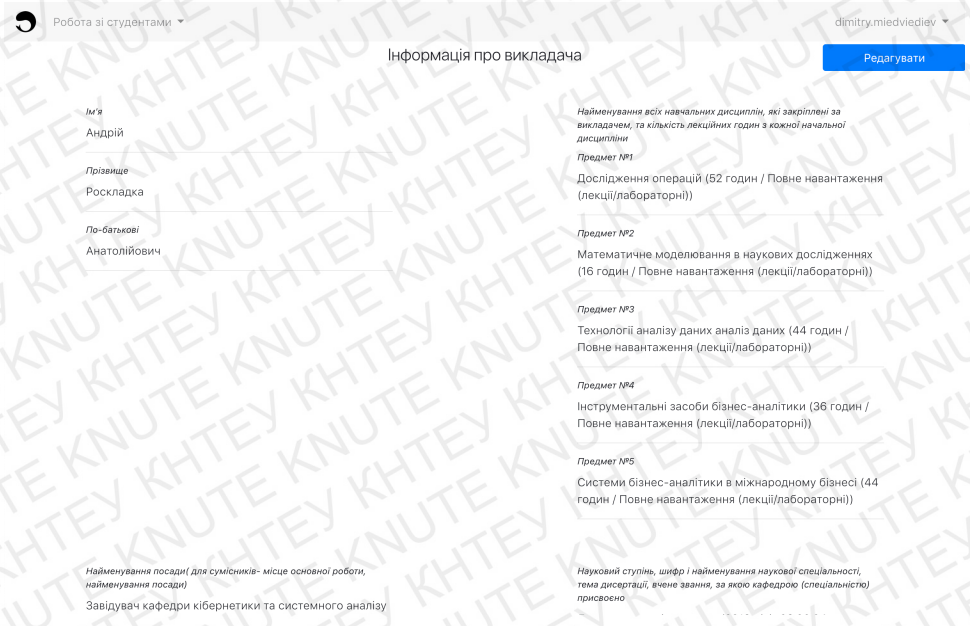

Рис 3.17 Профіль викладача

Джерело: [розроблено автором]

Рис 3.18 Сторінка створення викладача

Джерело: [розроблено автором]

Рис 3.19 Модуль роботи із списком кафедр

Джерело: [розроблено автором]

3.3. Тестування програмного продукту

Програміст повинен не тільки писати ефективні програми, але і знаходити в них помилки. Сучасна практика навчання програмуванню орієнтована, в основному, тільки на виконання програмістом першої половини своєї роботи.

Існують два типи програмних помилок:

- синтаксичні помилки[6,40,41] - виникають через порушення правил мови програмування. Такі помилки зазвичай виявляються під час компіляції. Можуть бути виключені порівняно легко. Навіть якщо не переглядати текст програми можна бути впевненим, що компілятор на стадії трансляції знайде помилки і видасть відповідні попередження. Фактично пошук помилок здійснює компілятор, а їхнє виправлення – програміст. Під час реалізації прототипу программи використовувався компілятор java-коду – IntellijIdea. Оскільки компілятор є доволі потужним з розширеним функціоналом і дозволяє завчасно правільно форматувати програмний код, при розробці вдалось уникнути багатьох проблем синтаксичного характеру. Також на уникнення синтаксичних помилок повпливало дотримання узгодження про найменування в рамках проекту.

- семантичні (логічні) помилки[6] - ті, що призводять до некоректних обчислень або помилок під час виконання (run-time error). Семантичні помилки усувають зазвичай за допомогою виконання програми з ретельно підібраними перевірочними даними, для яких відома правильна відповідь. Під час реалізації прототипу були проведені всі можливі попередні тести, щоб запобігти логічним помилкам, також неодноразово була перевірена логіка роботи прототипу у відповідності до бізнес-процессу.

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126. Тестування - це

процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок[4].

Якість не є абсолютною, адже це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів - це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури.

Тестування пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності програм заявленим вимогам.

Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціального додаткового тестування[40,6].

Існує безліч підходів до вирішення завдання тестування та верифікації ПЗ, але ефективне тестування складних програмних продуктів - це процес у вищій мірі творчий, не зводиться до прямування строгими і чіткими процедурами або до створення таких[4,6].

З точки зору ISO 9126 якість (програмних засобів) можна визначити як сукупну характеристику досліджуваного ПЗ з урахуванням наступних складових: надійність, супроводжуваність, практичність, ефективність, мобільність, функціональність.

Більш повний список атрибутів і критеріїв можна знайти в стандарті ISO 9126

Міжнародної організації зі стандартизації. Склад і зміст документації, супутньої процесу тестування, визначається стандартом IEEE 829-1998 Standard for Software Test Documentation.

Існує кілька ознак, за якими прийнято робити класифікацію видів тестування[2]. Зазвичай виділяють наступні:

За об'єктом тестування:

- функціональне тестування (functional testing) – проводилось у рамках розробки прототипу;
- тестування продуктивності (performance testing) – проводились попередні тести продуктивності на тестовому сервері:
- навантажувальне тестування (load testing) – проводилось попереднє навантажувальне тестування на тестовому сервері;
- стрес-тестування (stress testing) – в рамках розробки прототипу не проводилось;
- тестування стабільності (stability / endurance / soak testing) – проводилось в рамках розробки прототипу на тестовому сервері;
- тестування зручності використання (usability testing) – проводилось попереднє тестування;
- тестування інтерфейсу користувача (ui testing) – проводилось попереднє тестування;
- тестування безпеки (security testing) – не проводилось в рамках розробки прототипу;
- тестування локалізації (localization testing) – не проводилось;
- тестування сумісності (compatibility testing) – не проводилось оскільки веб-додатки не тестуються на сумісність з операційними системами.

За знанням системи:

- тестування чорного ящика (black box) – не проводилось;
- тестування білого ящика (white box) – не проводилось;

- тестування сірого ящика (gray box) – не проводилось.

За ступенем автоматизації:

- ручне тестування (manual testing) – проводилось під раз розробки прототипу;
- автоматизоване тестування (automated testing) – не проводилось, оскільки кількість логічного функціоналу програми недостатньо для обгрунтування проведення автоматизованого тестування;
- напівавтоматизоване тестування (semiautomated testing) – не проводилось.

За ступенем ізольованості компонентів:

- компонентне (модульне) тестування (component / unit testing) – проводилось під час розробки окремих блоків-модулів програмного продукту;
- інтеграційне тестування (integration testing) – проводилось в рамках тестування інтеграції системи та бази данних;
- системне тестування (system / end-to-end testing) – проводилось в межах розробки прототипу на фінальних стадіях.

За часом проведення тестування:

- Альфа-тестування (alpha testing):
  - тестування при прийманні (smoke testing);
  - тестування нової функціональності (new feature testing);
  - регресійне тестування (regression testing);
  - тестування при здачі (acceptance testing);
- Бета-тестування (beta testing).

За ознакою позитивності сценаріїв:

- позитивне тестування (positive testing);
- негативне тестування (negative testing).

За ступенем підготовленості до тестування:

- тестування за документацією (formal testing);
- інтуїтивне тестування (ad hoc testing).

У залежності від переслідуваних цілей види тестування можна умовно розділити на наступні типи:

- Функціональні.
- Нефункціональні.
- Пов'язані зі змінами.

Функціональні тести базуються на функціях та особливостях, а також на взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component/Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing)[6]. Функціональні види тестування розглядають зовнішню поведінку системи. Одні з найпоширеніших видів функціональних тестів:

- Функціональне тестування (Functional testing).
- Тестування безпеки (Security and Access Control Testing) - стратегія тестування, що використовується для перевірки безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту програми, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних. Тестування безпеки може виконуватися як автоматизовано так і в ручну, включаючи перевірку як позитивних, так і негативних тестових випадків. На момент розробки прототипу програмного продукту не було обгрунтованої потреби витрачати час на розробку програмних модулів, які відповідають за безпеку системи.
- Тестування взаємодії (Interoperability Testing) - це функціональне тестування, що перевіряє здатність програми взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності (compatibility testing) та інтеграційне тестування (integration testing). Фактично прототип автоматизованої системи обліку студентів інтегрований з кожним браурезом, які є.
- Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. У цілому, це тестування того, "Як" система працює.

Далі перераховані основні види нефукнціональних тестів:

- Тестування продуктивності[12,17]:
  - тестування навантаження (Performance and Load Testing) - визначення масштабованості додатків під навантаженням, при цьому відбувається:
    - вимір часу виконання вибраних операцій за певних інтенсивностей виконання цих операцій;
    - визначення кількості користувачів, що одночасно працюють з додатком;
    - визначення меж прийнятної продуктивності при збільшенні навантаження (при збільшенні інтенсивності виконання цих операцій);
    - дослідження продуктивності при високих, граничних, стресових навантаженнях;
  - стресове тестування (Stress Testing) дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу і також оцінити здатність системи до регенерації, тобто до повернення до нормального стану після припинення впливу стресу. Стресом у даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійна зміна конфігурації сервера. Також одним із завдань при стресовому тестуванні може бути оцінка деградації продуктивності, таким чином цілі стресового тестування можуть перетинатися з цілями тестування продуктивності;
  - тестування стабільності або надійності (Stability / Reliability Testing) - перевірка працездатності програми при тривалому (багатогодинному) тестуванні з середнім рівнем навантаження. Час виконання операцій може грати в даному виді тестування другорядну роль. При цьому на перше місце виходить відсутність витоків пам'яті, перезапусків серверів під навантаженням й інші аспекти, які впливають саме на стабільність роботи;

- об'ємне тестування (Volume Testing) - отримання оцінки продуктивності при збільшенні обсягів даних у базі даних програми, при цьому відбувається: вимір часу виконання вибраних операцій за певних інтенсивностей виконання цих операцій; може проводитися визначення кількості користувачів, що одночасно працюють з додатком;
- Тестування установки (Installation testing) спрямоване на перевірку успішної інсталяції та настройки, а також оновлення або видалення програмного забезпечення. На даний момент найбільш поширена установка ПЗ за допомогою інсталяторів (спеціальних програм, які самі по собі так само потребують належного тестування). У реальних умовах інсталяторів може не бути. У цьому випадку доведеться самостійно виконувати установку програмного забезпечення, використовуючи документацію у вигляді інструкцій або readme файлів, де крок за кроком описано всі необхідні дії та перевірки;

- тестування зручності користування (Usability Testing) - це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості та привабливості для користувачів розроблюваного продукту в контексті заданих умов. Тестування зручності користування дає оцінку рівня зручності використання програми за наступними пунктами:
  - продуктивність, ефективність (efficiency) - скільки часу і кроків знадобиться користувачеві для завершення основних завдань програми, наприклад, розміщення новини, реєстрації, покупки тощо (менше - краще);
  - правильність (accuracy) - скільки помилок зробив користувач під час роботи з додатком (менше - краще);
  - активізація в пам'яті (recall) - як багато користувач пам'ятає про роботу програми після припинення роботи з нею на тривалий період часу (повторне виконання операцій після перерви має проходити швидше ніж у нового користувача);

- емоційна реакція (emotional response) - як користувач почувається після завершення завдання - розгублений, знаходиться у стані стресу? Чи порекомендує користувач систему своїм друзям (позитивна реакція - краще)?

- Тестування на відмову і відновлення (Failover and Recovery Testing) перевіряє тестований продукт з точки зору здатності протистояти й успішно відновлюватися після можливих збоїв, що виникли у зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку (наприклад, відмова мережі)[23]. Метою даного виду тестування є перевірка систем відновлення (або дублюючих основний функціонал систем), які, у разі виникнення збоїв, забезпечать збереження і цілісність даних тестованого продукту.

- Конфігураційне тестування (Configuration Testing) - ще один вид традиційного тестування продуктивності. У цьому випадку замість того, щоб тестувати продуктивність системи з точки зору навантаження, тестується ефект впливу на продуктивність змін у конфігурації. Прикладом такого тестування можуть бути експерименти з різними методами балансування навантаження. Конфігураційне тестування також може бути поєднане з навантажувальним, стрес- або тестуванням стабільності[6].

Після проведення необхідних змін, таких як виправлення бага/дефекту, програмне забезпечення повинне бути перетестовано для підтвердження того факту, що проблему було дійсно вирішено. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

- Регресійне тестування (Regression Testing) - вид тестування спрямований на перевірку змін, зроблених у додатку або навколишньому середовищі (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб-сервер або сервер додатка), для підтвердження того факту, що існуюча раніше функціональність працює як і раніше.

- Тестування збірки (Build Verification Test) спрямоване на визначення відповідності випущеної версії критеріям якості для початку тестування. За своєю метою є аналогом димового тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглиб воно може проникати далі, в залежності від вимог до якості випущеної версії.
- Модульне тестування[6] (юніт-тестування) - тестується мінімально можливий для тестування компонент, наприклад, окремий клас або функція. Часто модульне тестування здійснюється розробниками ПЗ.
- Інтеграційне тестування[12] - тестуються інтерфейси між компонентами, підсистемами. За наявності резерву часу на даній стадії тестування ведеться ітераційно, з поступовим підключенням наступних підсистем.
- Рівні інтеграційного тестування:
    - компонентний інтеграційний рівень (Component Integration testing). Перевіряється взаємодія між компонентами системи після проведення компонентного тестування;
    - системний інтеграційний рівень (System Integration Testing). Перевіряється взаємодія між різними системами після проведення системного тестування.

Підходи до інтеграційного тестування:

- Знизу вгору (Bottom Up Integration).
- Зверху вниз (Top Down Integration)
- Великий вибух ("Big Bang" Integration)
- Системне тестування.
- Альфа-тестування.
- Бета-тестування
- Тестування "білого ящика" і "чорного ящика".
- Статичне і динамічне тестування.

- Покриття коду.
- Тестові скрипти.
- Приймальне тестування (Acceptance Testing).

Знизу вгору (Bottom Up Integration). Усі низькорівневі модулі, процедури або функції збираються воєдино і потім тестуються. Після чого збирається наступний рівень модулів для проведення інтеграційного тестування[6]. Даний підхід вважається корисним, якщо всі або практично всі модулі розроблюваного рівня готові. Також даний підхід допомагає визначити за результатами тестування рівень готовності додатків;

Зверху вниз (Top Down Integration). У першу чергу тестуються компоненти верхнього рівня ієрархії об'єктів з використанням заглушок замість компонентів більш низького рівня;

Великий вибух ("Big Bang" Integration). Усі або практично усі розроблені модулі збираються разом у вигляді закінченої системи або її основної частини, і потім проводиться інтеграційне тестування. Такий підхід дуже хороший для збереження часу[2]. Проте, якщо тест кейси та їх результати записані не вірно, то сам процес інтеграції дуже ускладниться, що стане перепоною для команди тестування при досягненні основної мети інтеграційного тестування.

Альфа-тестування - імітація реальної роботи з системою штатними розробниками, або реальна робота з системою потенційними користувачами / замовником. Найчастіше альфа-тестування проводиться на ранній стадії розробки продукту, але у деяких випадках може застосовуватися для закінченого продукту в якості внутрішнього приймального тестування. Іноді альфа-тестування виконується під відлагоджувачем або з використанням оточення, яке допомагає швидко виявляти знайдені помилки.

Бета-тестування - у деяких випадках виконується поширення версії з обмеженнями (за функціональністю або часом роботи) для певної групи осіб, з тим щоб переконатися, що продукт містить достатньо мало помилок.

Часто для вільного/відкритого ПЗ стадія альфа-тестування характеризує функціональне наповнення коду, а бета-тестування - стадію виправлення помилок. При цьому, як правило, на кожному етапі розробки проміжні результати роботи доступні кінцевим користувачам.

У термінології професіоналів[6] тестування фрази "тестування білого ящика" і "тестування чорного ящика" відносяться до того, чи має розробник тестів доступ до вихідного коду ПЗ, що тестується, або ж тестування виконується через інтерфейс користувача або прикладний програмний інтерфейс, наданий модулем, що тестується.

Під час тестування білого ящика (англ. white-box testing, також - прозорого ящика), розробник тесту має доступ до вихідного коду програм і може писати код, який пов'язаний з бібліотеками ПЗ, що тестується. Це типово для юніт-тестування (unit testing), при якому тестуються тільки окремі частини системи. Воно забезпечує те, що компоненти конструкції - працездатні й стійкі до певного ступеня. Під час тестування білого ящика використовуються метрики покриття коду.

При тестуванні чорного ящика тестер має доступ до ПЗ тільки через ті ж інтерфейси, що й замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування. Наприклад, модуль, що тестується[25], може віртуально натискати клавіші або кнопки миші за допомогою механізму взаємодії процесів, із упевненістю в тому, чи все йде правильно, що ці події викликають той же відгук, що й реальні натискання клавіш і кнопок миші. Як правило, тестування чорного ящика ведеться з використанням специфікацій чи інших документів, що описують вимоги до системи. В даному виді тестування критерій покриття складається з покриття структури вхідних даних, покриття вимог і покриття моделі (у тестуванні на основі моделей).

При тестуванні сірого ящика розробник тесту має доступ до вихідного коду, але при безпосередньому виконанні тестів доступ до коду, як правило, не потрібний.

Описані вище техніки - тестування білого ящика і тестування чорного ящика - припускають, що код виконується, і різниця полягає лише в тій інформації, якою

володіє тестувальник. В обох випадках це динамічне тестування.

При статичному тестуванні програмний код не виконується - аналіз програми відбувається на основі вихідного коду, який вираховується вручну, або аналізується спеціальними інструментами. У деяких випадках, аналізується не вихідний, а проміжний код (такий як байт-код або код на MSIL).

Також до статичного тестування відносять тестування вимог, специфікацій, документації.

Тестові скрипти. Тестувальники використовують тестові скрипти на різних рівнях: як у модульному, так і в інтеграційному і системному тестуванні. Тестові скрипти, як правило, пишуться для перевірки компонентів, в яких найбільш висока ймовірність появи відмов або вчасно не знайдена помилка може бути дорогою.

Покриття коду, за своєю суттю, є тестуванням методом білого ящика[37]. Протестоване ПЗ збирається зі спеціальними настройками або бібліотеками та/або запускається в особливому оточенні, в результаті чого для кожної використовуваної (виконуваної) функції програми визначається місцезнаходження цієї функції у вихідному коді. Цей процес дозволяє розробникам і фахівцям із забезпечення якості визначити частини системи, які, при нормальній роботі, використовуються дуже рідко або ніколи не використовуються (такі як код обробки помилок тощо). Це дозволяє зорієнтувати тестувальників на тестування найбільш важливих режимів.

Як правило, інструменти й бібліотеки, які використовуються для отримання покриття коду, вимагають значних витрат продуктивності та/або пам'яті, неприпустимих при нормальному функціонуванні ПЗ. Тому вони можуть використовуватися тільки в лабораторних умовах.

Приймальне тестування (Acceptance Testing) - формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться з метою: визначення чи задовольняє система приймальним критеріям; винесення рішення замовником або іншою уповноваженою особою приймається додаток чи ні[39].

Приймальне тестування виконується на підставі набору типових тестових

випадків і сценаріїв, розроблених на підставі вимог до даного додатку. Рішення про проведення приймального тестування приймається тоді, коли: продукт досяг необхідного рівня якості; замовник ознайомлений з Планом приймальних Робіт (Product Acceptance Plan) [40,39] або іншим документом, де описаний набір дій, пов'язаних з проведенням приймального тестування, дата проведення, відповідальні тощо.

Фаза приймального тестування триває до тих пір, поки замовник не виносить рішення про відправлення програми на доопрацювання або видачі додатку.

Висновки до розділу 3

Архітектура програмного забезпечення (англ. software architecture) - це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін стосується також документування архітектури програмного забезпечення. Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами, дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах.

Основною архітектури програмування є ідея зниження складності системи шляхом абстракції і розмежування повноважень.

Важливий аспект архітектури – це не тільки кінцевий результат, тобто сама архітектура, а й її логічне обґрунтування. Таким чином, важливо забезпечити документування рішень, які привели до створення цієї архітектури, і логічні обґрунтування таких рішень.

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може

містити більше технічних вимог, які описані в стандарті ISO 9126. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок.

Якість продукту не є абсолютною, адже це суб'єктивне поняття[39,19]. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує безліч підходів до вирішення завдання тестування та верифікації ПЗ, але ефективне тестування складних програмних продуктів - це процес у вищій мірі творчий і не зводиться до прямування строгими і чіткими процедурами.

ВИСНОВКИ

1. В процессі виконання дипломної роботи мною був проаналізований сучасни стан автоматизації ЗВО зокрема в Україні, методології та підходи що можна використати для розробки автоматизованої системи обліку студентів.

Сучасний досвід використання інформаційних систем автоматизації навчального процессу є, переважно, у закордонних ЗВО, які мають потужну фінансову базу для розгортання і використання подібних проєктів (систем) у своїх процесах.

В Україні найяскравішим і найвідомішим представником автоматизації процесів у сфері освіти - є ЄДЕБО, єдина державна електронна база з питань освіти, яка підтримується і фінансується Міністерством освіти і науки України.

Існує доволі багато підходів до розробки програмного забезпечення, одним з яких є методологія гнучкої розробки програмного забезпечення. Це клас методологій що базується на ітеративній розробці із залученням до створення вимог та реалізації розв'язків цих вимог залучають багатофункціональні команди здатні до самоорганізації, і який дозволяє підвищити продуктивність співробітників і, в той же час, мінімізувавши ризики.

Саме гнучкі методології привертають сьогодні увагу великих корпорацій за рахунок своєї результативності[19].

Для створення нової ефективної методології потрібно розуміти, що автоматизована система обліку студентів - це програмне забезпечення, що може базуватися на одному або декількох підходах до створення кінцевого продукту[26].

Специфіка цільової системи вимагає різних підходів до його розробки і оперативної підтримки. Сама методологія формується під впливом таких факторів як мета розробки, досвіду розробників, бюджету проекту тощо.

2. Були проаналізовані задачі і цілі створення проекту. Для успішної реалізації проекту «Автоматизована система обліку студентів» необхідно, у першу чергу,

визначити цілі та задачі проекту. Цей крок дозволить визначити, який результат ми отримаємо після завершення проекту, і які дії нам необхідно виконати для досягнення цієї цілі. Цілі і задачі мають бути правильно поставлені і бути чіткими заявами про наміри. Кожна ціль має мати власний напрям, який впливає на кінцевий результат усього проекту. Цілі і задачі мають бути вимірюваними[22].

Послідовне виконання завдань проекту дозволить якісно та швидко виконати необхідні 80 відсотків роботи над проектом, і створити функціональний прототип, який можна віддати на тестування кінцевому користувачу.

3. Були розглянуті підходи до створення моделей бізнес-процессів та діаграм діяльності для подібних проектів.

Важливою частиною побудови будь-якого проекту – є розуміння бізнес процесів, які необхідно реалізувати і автоматизувати у програмному забезпеченні.

Під бізнес процесом розуміється набір логічно взаємопов’язаних дій або завдань, виконання яких призводить до очікуваного результату. Тому, практично всі процеси організації можна віднести до бізнес процесів[27].

Основна мета управління бізнес процесами полягає у приведенні процесів у відповідність з цілями організації. Кожен процес повинен бути налаштований таким чином, щоб результати процесу приводили до досягнення бізнес цілей.

Процеси управління. Ці процеси призначені для планування, моніторингу та аналізу роботи. За рахунок процесів управління можна гарантувати досягнення цілей виробничими і процесами забезпечення. Процеси управління не додають цінності для кінцевого споживача, але вони необхідні для результативної та ефективної роботи організації. До таких процесів часто відносять процеси планування, постановки цілей, моніторингу та вимірювань, бюджетування та ін.

Виробничі (основні) процеси. За рахунок цих бізнес процесів організація досягає своїх цілей. Виробничі процеси забезпечують перетворення продукту або послуги і додають цінності для кінцевого споживача. До виробничих процесів відносять процеси проектування, виготовлення, надання послуг, монтажу та ін.

Моделювання бізнес-процесів дозволяє проаналізувати не тільки, як працює підприємство в цілому, як воно взаємодіє із зовнішніми організаціями, замовниками та постачальниками, але і як організована діяльність на кожному окремо взятому робочому місці.

Ще одним важливим моментом, який дозволяє краще підготуватись до реалізації проекту є діаграма діяльності, яка є візуальним представленням графу діяльності. Граф діяльності є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Дія є фундаментальною одиницею визначення поведінки в специфікації[14,13]. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів. Подібна специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

4. Були розроблені архітектура, макет та функціональний прототип проекту.

Однією з найголовніших задач – є побудова технічного завдання, вихідного документу для розробки нового програмного забезпечення (у даному випадку), в якому формуються основні цілі розробки, список принципових вимог до продукту, визначаються терміни та етапи розробки і регламентується процес приймально-здавальних випробувань. Цей документ містить основні вимоги замовника, вихідні дані для розробки, в ТЗ вказуються призначення продукту, область його застосування, стадії розробки різної документації, її склад, терміни виконання тощо, а також особливі вимоги, зумовлені специфікою проекту або умовами його експлуатації.

Архітектура програмного забезпечення (англ. software architecture) - це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін стосується також документування архітектури програмного забезпечення. Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами, дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий

дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах[10,41].

Основною архітектури програмування є ідея зниження складності системи шляхом абстракції і розмежування повноважень. Важливий аспект архітектури – це не тільки кінцевий результат, тобто сама архітектура, а й її логічне обґрунтування. Таким чином, важливо забезпечити документування рішень, які привели до створення цієї архітектури, і логічні обґрунтування таких рішень[38].

5. Було проведено тестування функціонального прототипу проекту.

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог, які описані в стандарті ISO 9126. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок.

Якість продукту не є абсолютною, адже це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує безліч підходів до вирішення завдання тестування та верифікації ПЗ, але ефективне тестування складних програмних продуктів - це процес у вищій мірі творчий, не зводиться до прямування строгими і чіткими процедурами або до створення таких.

Також були досягнуті всі поставлені задачі у повному обсязі. Додаток був розроблений для впровадження в робочий процесс обліку КНТЕУ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Архітектура програмного забезпечення / Software Engineering Institute // Web-сайт Інституту розробки програмного забезпечення (SEI) [Електронний ресурс] – Режим доступу: https://www.sei.cmu.edu/education-outreach/courses/course.cfm?coursecode=p35

2. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems / IEEE Computer Society // [Електронний ресурс] – режим доступу: http://cabibbo.dia.uniroma3.it/ids/altrui/ieee1471.pdf.

3. UML 2.0 Infrastructure Specification / Object Management Group, Inc. (OMG) // Unified Modeling Language (UML) Specification: Infrastructure version 2.0 [Електронний ресурс] – режим доступу: https://www2.informatik.hu-berlin.de/sam/lehre/MDA-UML/UML-Infra-03-09-15.pdf

4. Philippe Kruchten, The Rational Unified Process: An Introduction, Third Edition / Philippe Kruchten // Pearson Education Inc., 2004 – С.53 – 78

5. Philippe Kruchten, The Rational Unified Process: An Introduction, Third Edition / Philippe Kruchten // Pearson Education Inc., 2004 – С.141 – 156

6. Philippe Kruchten, The Rational Unified Process: An Introduction, Third Edition / Philippe Kruchten // Pearson Education Inc., 2004 – С.197 – 213

7. Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Second Edition / Len Bass, Paul Clements, and Rick Kazman // Addison Wesley, 2008 – Розділ 7.2

8. Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Second Edition / Len Bass, Paul Clements, and Rick Kazman // Addison Wesley, 2008 – Розділ 9.6

9. Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Second Edition / Len Bass, Paul Clements, and Rick Kazman // Addison Wesley, 2008 – Розділ 10.3

10. Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Second Edition / Len Bass, Paul Clements, and Rick Kazman // Addison Wesley, 2008 – Розділ 13.3

11. Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, Second Edition / Len Bass, Paul Clements, and Rick Kazman // Addison Wesley, 2008 – Розділ 16.3

12. James McGovern, et al., A Practical Guide to Enterprise Architecture / James McGovern // Prentice Hall, 2004 – С. 184-196

13. Grady Booch, James Rumbaugh, and Ivar Jacobson, The Unified Modeling Language User Guide / Grady Booch, James Rumbaugh, and Ivar Jacobson // Addison Wesley, 2008 – С.89-101

14. Grady Booch, James Rumbaugh, and Ivar Jacobson, The Unified Modeling Language User Guide / Grady Booch, James Rumbaugh, and Ivar Jacobson // Addison Wesley, 2008 – С.267-284

15. Grady Booch, James Rumbaugh, and Ivar Jacobson, The Unified Modeling Language User Guide / Grady Booch, James Rumbaugh, and Ivar Jacobson // Addison Wesley, 2008 – С.75-88

16. Muhammad Ali Babar, Torgeir Dingsøyr, Patricia Lago, Hans van der Vliet - Software Architecture Knowledge Management: Theory and Practice / Muhammad Ali Babar, Torgeir Dingsøyr, Patricia Lago, Hans van der Vliet / Springer Dordrecht Heidelberg 2009 – С. 40-44

17. Muhammad Ali Babar, Torgeir Dingsøyr, Patricia Lago, Hans van der Vliet - Software Architecture Knowledge Management: Theory and Practice / Muhammad Ali Babar, Torgeir Dingsøyr, Patricia Lago, Hans van der Vliet / Springer Dordrecht Heidelberg 2009 – С. 156-160

18. Unified Modeling Language Superstructure Specification V2.4.1 / Object Management Group, Inc. (OMG) // Unified Modeling Language (UML) Superstructure Specification: V2.4.1 [Електронний ресурс] – режим доступу:

https://www.omg.org/spec/UML/2.4.1/About-UML/UML/2.4.1/Superstructure/PDF/changebar

19. Кент Бек, Майк Бідл, Арі ван Беннекум, Алістер Кокбьорн, Вард Каннінгам, Мартін Фаулер, Джеймс Ґріннінґ, Джім Хайсміт, Енді Хант, Рон Джефріс, Джон Керн, Браян Марік, Роберт Мартін, Стів Меллор, Кен Щваббер, Джеф Сазерленд та Дейв Томас “Agile Manifesto” / “Маніфест Гнучкої Розробки” / Редакція 2001 [Електронний ресурс] – режим доступу: https://agilemanifesto.org

20. Kent Beck, Cynthia Andres Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series) / Kent Beck, Cynthia Andres // Addison Wesley, 2004 – С.37-51

21. Kent Beck, Cynthia Andres Extreme Programming Explained: Embrace Change, 2nd Edition (The XP Series) / Kent Beck, Cynthia Andres // Addison Wesley, 2004 – С.123-125

22. Alan Moran Managing Agile: Strategy, Implementation, Organisation and People / Alan Moran // Springer Dordrecht Heidelberg, 2015 – Розділ 4.1

23. Alan Moran Managing Agile: Strategy, Implementation, Organisation and People / Alan Moran // Springer Dordrecht Heidelberg, 2015 – Розділ 6.2

24. Alan Moran Managing Agile: Strategy, Implementation, Organisation and People / Alan Moran // Springer Dordrecht Heidelberg, 2015 – Розділ 8.3

25. Alan Moran Managing Agile: Strategy, Implementation, Organisation and People / Alan Moran // Springer Dordrecht Heidelberg, 2015 – Розділ 4.2.4

26. Alan Moran Managing Agile: Strategy, Implementation, Organisation and People / Alan Moran // Springer Dordrecht Heidelberg, 2015 – Розділ 3.2

27. Marianne Bradfort - Modern ERP: Select, Implement, and Use Today's Advanced Business Systems / Marianne Bradfort // North Carolina State University, College of Management, 2010 – Розділ 3.5

28. Marianne Bradfort - Modern ERP: Select, Implement, and Use Today's Advanced Business Systems / Marianne Bradfort // North Carolina State University, College of

Management, 2010 – Розділ 8.6

29. Marianne Bradfort - Modern ERP: Select, Implement, and Use Today's Advanced Business Systems / Marianne Bradfort // North Carolina State University, College of Management, 2010 – Розділ 5.3

30. Herbert Shildt., Java: The Complete Reference, Tenth Edition / Herbert Shildt // Oracle Press. – 2017, C.37-42

31. Herbert Shildt., Java: The Complete Reference, Tenth Edition / Herbert Shildt // Oracle Press. – 2017, C.111-117

32. Herbert Shildt., Java: The Complete Reference, Tenth Edition / Herbert Shildt // Oracle Press. – 2017, C.527-539

33. Robert Lafore., Data Structures and Algorithms in Java (2nd Edition) / Robert Lafore // Sams Publishing 2016, Розділ 3.2

34. Robert Lafore., Data Structures and Algorithms in Java (2nd Edition) / Robert Lafore // Sams Publishing 2016, Розділ 5.3

35. Robert Lafore., Data Structures and Algorithms in Java (2nd Edition) / Robert Lafore // Sams Publishing 2016, Розділ 11.4

36. Stephen Denning., The Age of Agile: How Smart Companies Are Transforming the Way Work Gets Done" / Stephen Denning // AMACOM, 2018 – Розділ 5.12

37. Stephen Denning., The Age of Agile: How Smart Companies Are Transforming the Way Work Gets Done" / Stephen Denning // AMACOM, 2018 – Розділ 11.1

38. Stephen Denning., The Age of Agile: How Smart Companies Are Transforming the Way Work Gets Done" / Stephen Denning // AMACOM, 2018 – Розділ 6.1

39. Jill Dyché, The CRM Handbook: A Business Guide to Customer Relationship Management / Jill Dyché // Addison Wesley, 2001 – C.102-117

40. Jill Dyché, The CRM Handbook: A Business Guide to Customer Relationship Management / Jill Dyché // Addison Wesley, 2001 – C.233-253

41. Jill Dyché, The CRM Handbook: A Business Guide to Customer Relationship Management / Jill Dyché // Addison Wesley, 2001 – C.267-270

42. Складові ERP-системи / режим доступу: https://searcherp.techtarget.com

43. Структура клієнт-серверної архітектури БД / режим доступу: https://searcherp.techtarget.com

44. Структура AGILE / режим доступу: https://codeproject.wordpress.com

45. Структура DSDM / режим доступу: https://en.wikipedia.org/wiki/Dynamic_systems_development_method

46. Структура XP / режим доступу: https://steemkr.com/technology/@samayyy/extreme-programming

47. Варіації можливих серверів / режим доступу: https://cloud.digitalocean.com