

Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з дисципліни
“Технологія створення програмних продуктів”
для студентів
напряму підготовки 6.050101
“Комп’ютерні науки”
(всіх форм навчання)

2014

Методичні вказівки до виконання лабораторних робіт з дисципліни “Технологія створення програмних продуктів” для студентів напряму підготовки 6.050101 “Комп’ютерні науки” (всіх форм навчання) / А. О. Олійник, Т. О. Колпакова, В. М. Льовкін. – Запоріжжя : ЗНТУ, 2012. – 61 с.

Автори: А. О. Олійник, к.т.н., доцент
 Т. О. Колпакова, асистент
 В. М. Льовкін, к.т.н., доцент

Рецензент: С. О. Субботін, д.т.н., професор

Відповідальний
за випуск: В. І. Дубровін, к.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “20” серпня 2014 р.

ЗМІСТ

Вступ	6
1 Лабораторна робота № 1 Збирання та аналіз вимог. Стислий опис проекту	7
1.1 Мета роботи	7
1.2 Короткі теоретичні відомості	7
1.2.1 Етапи аналізу вимог	7
1.2.2 Класифікація вимог	7
1.2.3 Документування вимог	8
1.3 Завдання на лабораторну роботу.....	11
1.4 Зміст звіту.....	12
1.5 Контрольні запитання	12
2 Лабораторна робота № 2 Розроблення технічного завдання на проєкт системи.....	13
2.1 Мета роботи	13
2.2 Короткі теоретичні відомості	13
2.2.1 Основне призначення ТЗ.....	13
2.2.2 Етапи складання списку вимог ТЗ	14
2.2.3 Структура ТЗ.....	14
2.3 Завдання на лабораторну роботу.....	15
2.4 Зміст звіту.....	15
2.5 Контрольні запитання	16
3 Лабораторна робота № 3 Проектування з використанням UML. Структурні діаграми.....	17
3.1 Мета роботи	17
3.2 Короткі теоретичні відомості	17
3.2.1 Діаграма класів	17
3.2.2 Діаграма пакетів	19
3.2.3 Діаграма компонентів	20
3.3 Завдання на лабораторну роботу.....	21
3.4 Зміст звіту.....	21
3.5 Контрольні запитання	22
4 Лабораторна робота № 4 Проектування з використанням UML. Діаграми поведінки.....	23

4.1 Мета роботи	23
4.2 Короткі теоретичні відомості	23
4.2.1 Діаграма прецедентів	23
4.2.2 Діаграма станів	24
4.2.3 Діаграма діяльності	25
4.3 Завдання на лабораторну роботу.....	27
4.4 Зміст звіту.....	27
4.5 Контрольні запитання	28

5 Лабораторна робота № 5 Проектування з використанням UML. Діаграми взаємодії.....29

5.1 Мета роботи	29
5.2 Короткі теоретичні відомості	29
5.2.1 Діаграма кооперації.....	29
5.2.2 Діаграма послідовності	30
5.3 Завдання на лабораторну роботу.....	32
5.4 Зміст звіту.....	32
5.5 Контрольні запитання	32

6 Лабораторна робота № 6 Аналіз архітектури веб-додатків на основі предметної області.....34

6.1 Мета роботи	34
6.2 Короткі теоретичні відомості	34
6.3 Завдання на лабораторну роботу.....	35
6.4 Зміст звіту.....	35
6.5 Контрольні запитання	35

7 Лабораторна робота № 7 Аналіз архітектури веб-додатків з використанням CMS.....36

7.1 Мета роботи	36
7.2 Короткі теоретичні відомості	36
7.3 Завдання на лабораторну роботу.....	38
7.4 Зміст звіту.....	38
7.5 Контрольні запитання	39

8 Лабораторна робота № 8 Проектування інтерфейсу веб-додатків.....40

8.1 Мета роботи	40
8.2 Короткі теоретичні відомості	40
8.2.1 Модель GOMS	40

8.2.2 Закон Фіттса	42
8.2.3 Закон Хіка.....	42
8.3 Завдання на лабораторну роботу.....	43
8.4 Зміст звіту.....	43
8.5 Контрольні запитання	44
9 Лабораторна робота № 9 Аналіз структури веб-додатків.....	45
9.1 Мета роботи	45
9.2 Короткі теоретичні відомості	45
9.2.1 Базові структурні компоненти CMS	45
9.2.2 Класифікація CMS	45
9.3 Завдання на лабораторну роботу.....	46
9.4 Зміст звіту.....	46
9.5 Контрольні запитання	47
10 Лабораторна робота № 10 Розгортання веб-додатків.....	48
10.1 Мета роботи	48
10.2 Короткі теоретичні відомості	48
10.3 Завдання на лабораторну роботу.....	50
10.4 Зміст звіту.....	50
10.5 Контрольні запитання	50
11 Лабораторна робота № 11 Аналіз якості та оцінка веб-додатків.....	51
11.1 Мета роботи	51
11.2 Короткі теоретичні відомості	51
11.3 Завдання на лабораторну роботу.....	54
11.4 Зміст звіту.....	55
11.5 Контрольні запитання	55
12 Лабораторна робота № 12 Розширення функціональності веб-додатків за допомогою модулів.....	56
12.1 Мета роботи	56
12.2 Короткі теоретичні відомості	56
12.2.1 Модульний принцип організації CMS	56
12.2.2 Приклади розширення функціональності CMS	57
12.3 Завдання на лабораторну роботу.....	59
12.4 Зміст звіту.....	60
12.5 Контрольні запитання	60
Література.....	61

ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами усіх форм навчання основ архітектури та проектування програмного забезпечення.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. Звичайно, в дані методичні вказівки неможливо було внести весь матеріал, необхідний для виконання та захисту лабораторних робіт. Тому тут містяться основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт. Таким чином для виконання лабораторної роботи та при підготовці до її захисту необхідно ознайомитись з конспектом лекцій та проробити весь матеріал, наведений в переліку рекомендованої літературі. При цьому не варто обмежуватись лише наведеним списком.

Для одержання заліку з кожної роботи студент здає викладачу цілком оформлений звіт, а також демонструє на екрані комп'ютера результати виконання лабораторної роботи.

Звіт має містити:

- титульний аркуш;
- тему та мету роботи;
- завдання до роботи;
- лаконічний опис теоретичних відомостей;
- результати виконання лабораторної роботи;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 × 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок або вміщують у канцелярський файл.

Під час співбесіди при захисті лабораторної роботи студент повинний виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів оформленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

1 ЛАБОРАТОРНА РОБОТА № 1 ЗБИРАННЯ ТА АНАЛІЗ ВИМОГ. СТИСЛИЙ ОПИС ПРОЕКТУ

1.1 Мета роботи

Навчитися виконувати збирання, аналіз та документування вимог до програмного забезпечення.

1.2 Короткі теоретичні відомості

Аналіз вимог полягає у визначенні потреб та умов, які висуваються щодо нового, чи зміненого продукту, враховуючи можливо конфліктні вимоги різних замовників.

Аналіз вимог є критичним для успішного розроблення проекту. Вимоги мають бути задокументованими, вимірними, тестованими, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи.

1.2.1 Етапи аналізу вимог

Аналіз вимог включає наступні види діяльності:

Виявлення (збирання) вимог – задача комунікації з користувачами для визначення їх вимог.

Аналіз вимог – виявлення недоліків вимог (неточностей, неповноти, неоднозначностей чи суперечностей) і їх виправлення.

Запис вимог – документування вимог в різних формах, таких як опис звичайною мовою, прецедентами, користувацькими історіями, чи специфікаціями процесу.

1.2.2 Класифікація вимог

Вимоги споживача представляють собою вирази фактів та припущень, які описують очікування від системи в термінах цілей, середовища, обмежень та міри ефективності й придатності.

Архітектурні вимоги пояснюють, що має бути зроблено ідентифікацією необхідної системної архітектури.

Структурні вимоги пояснюють, що має бути зроблено ідентифікацією необхідної структури системи.

Поведінкові вимоги пояснюють, що має бути зроблено ідентифікацією необхідної поведінки системи.

Функціональні вимоги пояснюють, що має бути зроблено ідентифікацією необхідної задачі, дії чи діяльності, які мають виконуватись. Аналіз функціональних вимог буде використаний в функціях верхніх рівнів для функціонального аналізу.

Нефункціональні вимоги задають критерій для оцінювання операцій системи, замість її поведінки.

Вимоги продуктивності пояснюють, до якої міри місії чи функції повинні бути виконані; зазвичай вимірюється в термінах кількості, якості, охоплення, своєчасності чи готовності. Протягом аналізу вимог, вимоги продуктивності будуть інтерактивно розроблятися впродовж всіх виявлених функцій, що базуються на факторах життєвого циклу системи, і характеризуються в термінах ступеня визначеності в їх оцінках, ступеня критичності успіху системи і їх відношення до інших вимог.

Вимоги дизайну.

Успадковані вимоги, тобто вимоги, які обумовлені вимогами вищого рівня, чи перетворені з них.

Розподілені вимоги визначені поділом чи іншим перерозміщенням високорівневих вимог в декілька низькорівневих вимог.

1.2.3 Документування вимог

Документ, який описує вимоги, є результатом етапів виявлення та аналізу вимог. Документ опису вимог розроблюється відповідно до раніше визначеного шаблону. Шаблон визначає структуру та стиль документу.

Частина документу опису вимог, яка містить **стислий опис проекту**, переважно орієнтує тих керівників та учасників проекту, які відповідні за прийняття рішення, але ймовірно не стануть детально вивчати документ повністю. На початку документу необхідно визначити цілі та межі проекту, а потім описати діловий контекст системи.

Необхідно визначити учасників проекту системи. При цьому важливо, щоб замовник виступав не як безвиразно представлений підрозділ або офіс, необхідно привести конкретні імена.

Хоча документ опису вимог може бути далеким від технічних рішень, важливо визначити ідеї, які стосуються рішення, на початкових етапах життєвого циклу розробки. Важливо також проаналізувати варіант придбання готового продукту замість його розроблення “з нуля”. Документ опису вимог повинен надавати перелік існуючих програмних компонентів та пакетів, які необхідно в подальшому вивчити в якості варіантів можливих рішень.

Основна частина документу опису вимог присвячена визначенню **системних сервісів**. Ця частина може займати до половини всього обсягу документу. Це єдина частина документу, яка може містити узагальнені моделі – моделі бізнес-вимог.

Межі системи можна моделювати за допомогою діаграм контексту. У поясненнях до діаграми контексту повинні бути чітко визначені межі системи. Без подібного визначення проект не може бути застрахованим від спроб “розтягнути” його межі.

Функціональні вимоги можна моделювати за допомогою діаграм бізнес-прецедентів. Однак, діаграми охоплюють перелік функціональних вимог тільки в загальному вигляді. Усі вимоги треба позначити, класифікувати та визначити.

Вимоги до даних можна моделювати за допомогою діаграми бізнес-класів. Так само як і у випадку функціональних вимог, діаграма бізнес-класів не дає повного визначення структур даних для бізнес-процесів. Кожний бізнес-клас вимагає подальших пояснень. Необхідно описати атрибут ненаповнення класів та визначити ідентифікуючі атрибути класів. У протилежному випадку неможливо правильно представити асоціації.

Системні сервіси визначають, що повинна робити система. Системні обмеження визначають, на скільки система обмежена під час виконання обслуговування. Системні обмеження пов’язані з наступними видами вимог:

- вимоги до інтерфейсу;
- вимоги до продуктивності;
- вимоги до безпеки;
- експлуатаційні вимоги;
- політичні та юридичні вимоги.

Вимоги до інтерфейсу визначають, як система взаємодіє з користувачем. У документі опису вимог визначаються тільки “відчуття”

від GUI-інтерфейсу. Початкове проектування GUI-інтерфейсу виконується під час специфікації вимог та пізніше під час системного проектування.

У залежності від галузі застосування вимоги до продуктивності можуть грати доволі важливу роль в успіху проекту. В обмеженому розумінні вони задають швидкість (час відгуку системи), з якою повинні виконуватися різноманітні завдання. У широкому розумінні вимоги до продуктивності включають інші обмеження: щодо надійності, готовності, пропускну здатності тощо.

Вимоги до безпеки описують користувацькі права доступу до інформації, що контролюються системою. Користувачам може бути наданий обмежений доступ до даних або обмежені права на виконання деяких операцій з даними.

Експлуатаційні вимоги визначають програмно-технічне середовище, якщо воно відоме на етапі проектування, у якому повинна функціонувати система. Ці вимоги можуть впливати на інші сторони проекту, такі як: підготовка користувачів та супроводження системи.

Важливі і інші види обмежень. Наприклад, у відношенні деяких систем можуть висуватися вимоги щодо легкості їх використання (вимоги щодо придатності їх використання) або легкості їх супроводження (вимоги щодо придатності до супроводження).

Значення визначення чітких системних обмежень важко переоцінити. Існує багато прикладів проектів, які провалилися через втрачені обмеження або такі обмеження, які були невірно зрозумілі. Дана проблема у рівній мірі стосується як замовників, так і розробників.

Заключна частина документу опису вимог визначає **інші проектні питання**. Одним з важливих розділів даної частини є “Відкриті питання”, у якому визначаються всі питання, які можуть вплинути на успіх проекту і які не розглядалися в інших розділах документу. До даного пункту належить очікуване збільшення значення деяких вимог, які в поточний момент виходять за межі проекту, а також будь-які потенційні проблеми та відхилення у поведінці системи, які можуть початися у зв’язку з розгортанням системи.

У даній частині документу опису вимог необхідно представити попередній план-графік виконання основних проектних завдань, а також попередній розподіл людських та інших ресурсів. Для вироблення стандартних планових графіків можна використовувати програмні засоби управління проектами, наприклад, такі як система PERT (program

evaluation-and-review technique – метод оцінювання та перегляду планів) або карти Ганта.

Прямим результатом складання план-графіку може бути розроблення попереднього бюджету. Вартість проекту може бути виражена у вигляді діапазонів значень витрат, а не конкретного значення.

Додатки до документу опису вимог містять іншу корисну для розуміння вимог інформацію. Основним додатком є глосарій. **Глосарій** визначає терміни, скорочення та аббревіатури, які використовуються в документі опису вимог. Значення вірно розробленого глосарію важко переоцінити. Невірне використання термінології несе велику небезпеку для проекту.

Розділ посилань містить перелік документів, які згадуються та використовуються при підготовці документу опису вимог. До них можуть належати книги та інші опубліковані джерела інформації, а також внутрішні документи, які можливо є навіть більш важливими.

1.3 Завдання на лабораторну роботу

1.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

1.3.2 Обрати тему для подальшого проектування системи.

1.3.3 Розробити загальну схему системи за обраною темою.

1.3.4 Виконати стислий опис проекту: сформулювати цілі та межі проекту, визначити учасників проекту, розглянути існуючі рішення проблеми.

1.3.5 Визначити системні сервіси: межі системи, функціональні вимоги, вимоги до даних.

1.3.6 Сформулювати системні обмеження: вимоги до інтерфейсу, продуктивності, безпеки, експлуатаційні вимоги.

1.3.7 Визначити проектні питання: представити попередній план-графік виконання основних проектних завдань, попередній бюджет, розглянути питання, які впливають на успіх проекту і не були розглянуті в інших пунктах.

1.3.8 Сформулювати додатки до документу опису вимог: глосарій, посилання.

1.3.9 Оформити звіт з роботи.

1.3.10 Відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Тема, обрана для проектування.

1.4.3 Загальна схема системи.

1.4.4 Завдання на лабораторну роботу.

1.4.5 Вимоги до програмного забезпечення: стислий опис проекту, системні сервіси, системні обмеження, проектні питання, додатки.

1.4.6 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

1.5 Контрольні запитання

1.5.1 Які етапи життєвого циклу розробки систем розрізняють?

1.5.2 Які існують методи виявлення вимог до програмного забезпечення?

1.5.3 Які фази включає процес аналізу вимог?

1.5.4 Кого можна віднести до осіб, зацікавлених у розробці системи?

1.5.5 Які існують способи документування вимог?

1.5.6 Які типи вимог можна виокремити?

1.5.7 Які види ризиків характерні для вимог?

1.5.8 Що таке глосарій?

1.5.9 Яка типова структура документу опису вимог?

2 ЛАБОРАТОРНА РОБОТА № 2 РОЗРОБЛЕННЯ ТЕХНІЧНОГО ЗАВДАННЯ НА ПРОЕКТ СИСТЕМИ

2.1 Мета роботи

Навчитися визначати функціональність системи для розроблення технічного завдання на проект системи.

2.2 Короткі теоретичні відомості

Технічне завдання (ТЗ) – вихідний документ для розроблення автоматизованої системи або створення програмного продукту, відповідно до якого проводиться виготовлення, приймання при введенні в дію та експлуатація відповідного об'єкта. ТЗ є основним документом, що визначає вимоги і порядок створення (розвитку або модернізації) інформаційної системи.

2.2.1 Основне призначення ТЗ

Як інструмент комунікації між замовником та виконавцем, ТЗ дозволяє обом сторонам:

- уявити готовий продукт;
- виконати перевірку готового продукту (приймальне тестування – проведення випробувань);
- зменшити кількість помилок, пов'язаних зі зміною вимог внаслідок їх неповноти або хибності.

Замовнику:

- усвідомити, що саме йому потрібно, спираючись на існуючі на даний момент технічні можливості і свої ресурси;
- вимагати від виконавця відповідності продукту всім умовам, обговореним в ТЗ.

Виконавцю:

- зрозуміти суть завдання, показати замовнику «технічний вигляд» майбутнього програмного продукту;
- спланувати виконання проекту;
- відмовитися від виконання робіт, не зазначених у ТЗ.

2.2.2 Етапи складання списку вимог ТЗ

Робота над ТЗ включає виконання низки етапів, а невизначеність, властива цій роботі, викликає проходження їх по декілька разів, ітераційно, від більш загальної постановки завдання до детального опрацювання (проектування носить ітераційний характер і те, що не враховано на початку, може бути враховано на наступних етапах).

Основні етапи розроблення ТЗ:

- аналіз завдання замовника;
- конкретизація цілей проектування;
- обробка зібраної інформації:
 - 1) узагальнення та абстрагування;
 - 2) перевірка на суперечливість;
 - 3) розмежування вимог на умови, обмеження та показники якості;
 - 4) параметризація;
 - 5) зменшення списку вимог;
 - 6) зведення вимог та затвердження замовником.

2.2.3 Структура ТЗ

Технічне завдання на проект системи має включати наступні пункти:

- вступ, де наводиться найменування та коротка характеристика області застосування програмного забезпечення (ПЗ);
- підстави для розроблення, де зазначаються умови для проведення розробки, найменування та умовне позначення теми розробки;
- призначення розробки, де необхідно визначити функціональне та експлуатаційне призначення;
- основні вимоги до ПЗ, що включають: вимоги до функціональних характеристик, вимоги до надійності, умови експлуатації, вимоги до складу та параметрів технічних засобів, вимоги до маркування та упакування, вимоги до транспортування та збереження, вимоги до програмної документації;
- стадії та етапи розробки;
- порядок контролю та приймання системи.

2.3 Завдання на лабораторну роботу

2.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

2.3.2 Визначити характеристику області застосування ПЗ, підстави розроблення системи та призначення розробки.

2.3.3 Сформулювати основні вимоги до ПЗ та до програмної документації.

2.3.4 Визначити випробування для контролю та приймання програмного продукту.

2.3.5 Сформулювати технічне завдання на проект системи за обраною темою.

2.3.6 Оформити звіт з роботи.

2.3.7 Відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Тема, обрана для проектування.

2.4.3 Загальна схема системи.

2.4.4 Технічне завдання на проект системи, яке має включати наступні пункти:

- вступ: найменування та коротка характеристика області застосування ПЗ;
- підстави для розроблення: умови для проведення розробки, найменування та умовне позначення теми розробки;
- призначення розробки: функціональне та експлуатаційне призначення;
- основні вимоги до ПЗ: вимоги до функціональних характеристик, вимоги до надійності, умови експлуатації, вимоги до складу та параметрів технічних засобів, вимоги до маркування та упакування, вимоги до транспортування та збереження, вимоги до програмної документації;
- стадії та етапи розробки;
- порядок контролю та приймання системи.

2.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

2.5 Контрольні запитання

2.5.1 Хто відповідає за розроблення технічного завдання на проєкт системи?

2.5.2 Яку інформацію надає технічне завдання виконавцям та замовнику?

2.5.3 Для чого необхідно розробляти технічне завдання?

2.5.4 Які етапи включає робота над технічним завданням?

2.5.5 Що таке функціональне призначення розробки?

2.5.6 Що таке експлуатаційне призначення розробки?

2.5.7 Які стандарти регламентують склад технічного завдання?

3 ЛАБОРАТОРНА РОБОТА № 3 ПРОЕКТУВАННЯ З ВИКОРИСТАННЯМ UML. СТРУКТУРНІ ДІАГРАМИ

3.1 Мета роботи

Ознайомитися з видами структурних діаграм та особливостями їх застосування, навчитися розробляти абстрактну модель системи мовою UML на основі структурних діаграм.

3.2 Короткі теоретичні відомості

UML (англ. Unified Modeling Language) – уніфікована мова моделювання, що використовується у парадигмі об'єктно-орієнтованого програмування та є невід'ємною частиною уніфікованого процесу розроблення ПЗ. UML заснована на діаграмах, що дають можливість представити систему у такому вигляді, щоб її можна було легко перевести в програмний код.

3.2.1 Діаграма класів

Діаграму класів (class diagram) використовують для подання статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування.

Класи використовуються для складання словника розроблюваної системи. Вони представляють опис сукупності об'єктів із загальними атрибутами, операціями, відношеннями та семантикою. Клас реалізує один чи декілька інтерфейсів.

Діаграми класів використовуються для моделювання статичного вигляду системи з точки зору проектування. Сюди більшою мірою належить моделювання словника системи, кооперацій та схем. Діаграми класів складають основу діаграм компонентів.

Діаграми класів можуть містити наступні сутності:

- класи;
- інтерфейси – набори операцій, які використовуються для специфікації послуг, що надаються класом або компонентом;

- кооперації – спільнота класів, інтерфейсів та інших елементів, які працюють спільно для забезпечення кооперативної поведінки, що є більш значимою ніж сума складових;
- відношення залежності (описує існуючі між класами відношення використання), узагальнення (зв'язує узагальнені класи зі спеціалізованими) та асоціації (надає структурні відношення між об'єктами).

Залежністю називають варіант використання, відповідно до якого зміна у специфікації одного елементу може вплинути на інший елемент, що його використовує, при чому зворотне не обов'язково. Графічно залежність позначається пунктирною лінією зі стрілкою, направленою від одного елементу на той, від якого він залежить. Найчастіше залежності застосовуються під час роботи з класами, щоб відбити у сигнатурі операції той факт, що один клас використовує інший у якості аргументу.

Узагальнення – відношення між загальною сутністю (суперкласом, або батьківським класом) та її конкретним втіленням (субкласом, чи нащадком). Узагальнення іноді називають відношенням типу “є”, маючи на увазі, що одна сутність є окремим вираженням іншої, більш загальної. Узагальнення означає, що об'єкти класу-нащадка можуть використовуватися всюди, де зустрічаються об'єкти батьківського класу, але не навпаки. Нащадок наслідуює властивості батьківського класу, зокрема його атрибути та операції.

Проста асоціація між двома класами відбиває структурне відношення між рівноправними сутностями, коли обидва класи знаходяться на одному концептуальному рівні і не один з них не є більш важливим ніж інший. Але іноді треба змодельовати відношення типу “частина/ціле”, в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин). Відношення такого типу називається агрегацією. Агрегація є окремим випадком асоціації.

Часто під час моделювання важливо зазначити, скільки об'єктів може бути пов'язано шляхом одного екземпляру асоціації. Це число називається кратністю ролі асоціації і записується або як вираз, значенням якого є діапазон значень, або у явному вигляді.

Приклад діаграми класів наведено на рис. 3.1.

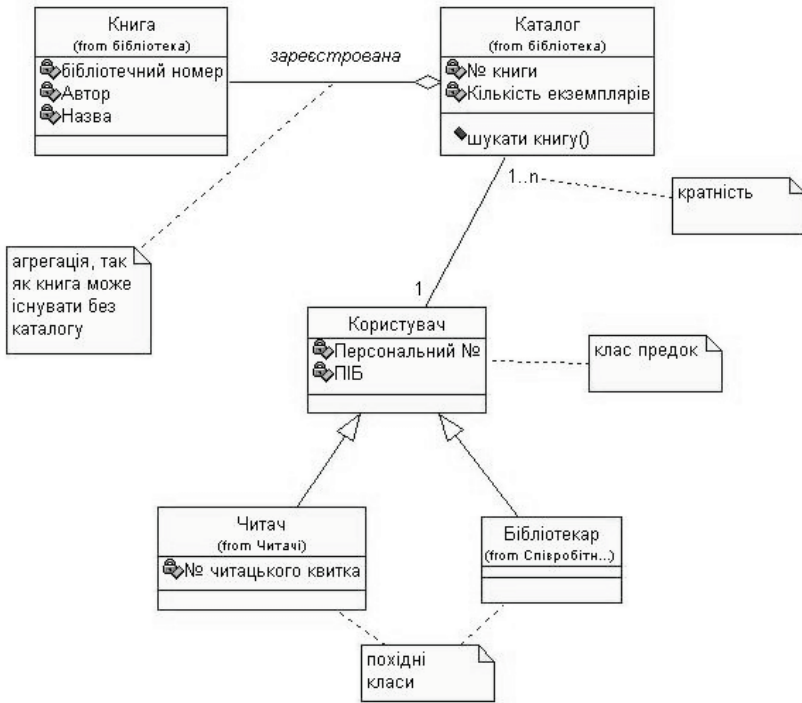


Рисунок 3.1 – Діаграма класів

3.2.2 Діаграма пакетів

Діаграма пакетів є різновидом діаграми класів, на якій відображаються тільки пакети і залежності.

Пакети використовуються для організації моделей та дозволяють колективу розробників ефективно ними маніпулювати під час спільної роботи. Пакети визначають простір імен для елементів моделі, які вони містять, і не несуть більш ніякої інформації.

Між двома пакетами існує деяка залежність, якщо існує яка-небудь залежність між будь-якими двома класами в пакетах.

Хоча пакети не дають відповіді на питання, як зменшити кількість залежностей у системі, проте вони допомагають виділити ці за-

лежності. Як тільки вони будуть визначені, залишиться лише попрацювати над їх скороченням.

Приклад діаграми пакетів наведено на рис. 3.2.

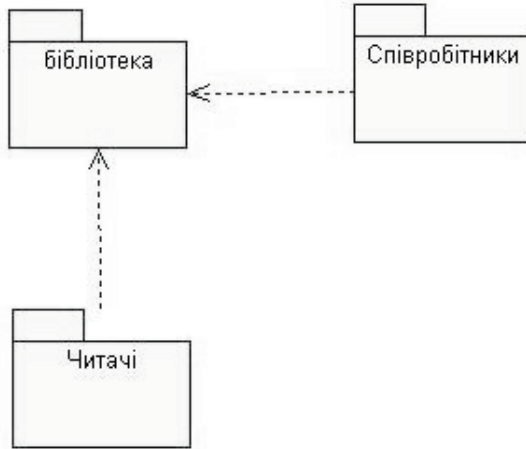


Рисунок 3.2 – Діаграма пакетів

3.2.3 Діаграма компонентів

Діаграма компонентів (рис. 3.3) описує особливості фізичної уяви системи. Дана діаграма застосовується для моделювання статичного вигляду системи з точки зору реалізації. За своєю сутністю діаграми компонентів – не що інше як діаграми класів, сфокусовані на системних компонентах.

Діаграма компонентів відображає залежності між компонентами ПЗ, включаючи компоненти вихідних кодів, бінарні компоненти та компоненти, що можуть виконуватись, тобто вона дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами.

Діаграми компонентів зазвичай включають:

- компоненти;
- інтерфейси;

– відношення узагальнення, асоціації та реалізації.

Реалізацією називається семантичне відношення між класифікаторами, за якого один з них описує контракт, а інший гарантує його виконання. Частіше за все реалізації використовуються для визначення відношень між інтерфейсом та класом або компонентом, який надає об'явлені в інтерфейсі операції або послуги.

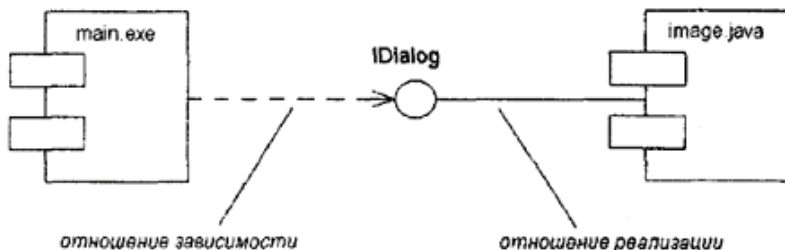


Рисунок 3.3 – Діаграма компонентів

3.3 Завдання на лабораторну роботу

3.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

3.3.2 Вивчити правила побудови структурних діаграм: діаграм класів, компонентів, розгортання, об'єктів, пакетів, профілів, кооперації.

3.3.3 Розробити абстрактну модель системи за обраною темою на основі діаграми класів.

3.3.4 Розробити абстрактну модель системи за обраною темою на основі діаграми пакетів.

3.3.5 Розробити абстрактну модель системи за обраною темою на основі діаграми компонентів.

3.3.6 Оформити звіт з роботи.

3.3.7 Відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Тема, обрана для проектування.

3.4.3 Загальна схема роботи системи.

3.4.4 Діаграма класів системи, що проектується.

3.4.5 Діаграма компонентів системи, що проектується.

3.4.6 Діаграма пакетів системи, що проектується.

3.4.7 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

3.5 Контрольні запитання

3.5.1 Які види структурних діаграм розрізняють в UML?

3.5.2 З якою метою використовуються діаграми класів?

3.5.3 Що таке структурні класи?

3.5.4 Які види зв'язків між сутностями на діаграмах класів представлені в UML?

3.5.5 Який вид відношень між класами позначають асоціації і які види асоціацій розрізняють?

3.5.6 Відношення узагальнення між класами належить до часу проектування чи до часу виконання?

3.5.7 Які типи відношення залежності між класами розрізняють і яке вони мають значення?

3.5.8 Яку інформацію несуть діаграми компонентів?

3.5.9 З яких точок зору розглядається система в діаграмах класів, об'єктів та компонентів?

3.5.10 Для чого застосовуються діаграми компонентів?

3.5.11 У яких випадках використовуються діаграми розгортання?

3.5.12 Які елементи включає в себе діаграма розгортання?

3.5.13 Які види залежностей використовуються в діаграмах пакетів?

3.5.14 Які види структурних діаграм можуть бути застосовані для моделювання фізичних баз даних?

4 ЛАБОРАТОРНА РОБОТА № 4 ПРОЕКТУВАННЯ З ВИКОРИСТАННЯМ UML. ДІАГРАМИ ПОВЕДІНКИ

4.1 Мета роботи

Ознайомитися з видами діаграм поведінки та особливостями їх застосування, навчитися розробляти абстрактну модель системи мовою UML на основі діаграм поведінки.

4.2 Короткі теоретичні відомості

4.2.1 Діаграма прецедентів

Діаграми прецедентів застосовуються для моделювання вигляду системи з точки зору прецедентів (або варіантів використання). Даний вид діаграм використовується для моделювання динамічних аспектів системи, так само як і діаграми станів та діяльності.

Діаграма прецедентів – діаграма, на якій зображено відношення між акторами та прецедентами в системі (рис. 4.1).

Діаграми прецедентів зазвичай включають в себе:

- прецеденти;
- акторів;
- відношення залежності, узагальнення та асоціації.

UML дозволяє моделювати контекст за допомогою прецедентів, у яких особлива увага акцентується акторів, що оточують систему. Важливо вірно визначити акторів, оскільки це дозволяє описати клас сутностей, що взаємодіють з системою.

Моделювання контексту системи складається з наступних кроків:

а) ідентифікуйте акторів, що оточують систему: визначіть групи, яким участь системи необхідна для виконання їхніх завдань; групи, які необхідні для виконання системою своїх функцій; групи, що взаємодіють із зовнішніми програмними та апаратними засобами, а також групи, що виконують допоміжні функції адміністрування та підтримки;

б) організуйте схожих акторів за допомогою відношень узагальнення/спеціалізації;

в) введіть стереотипи для кожного актору, якщо це полегшує розуміння;

г) розташуйте акторів на діаграмі прецедентів та визначіть способи їх зв'язку з прецедентами системи.

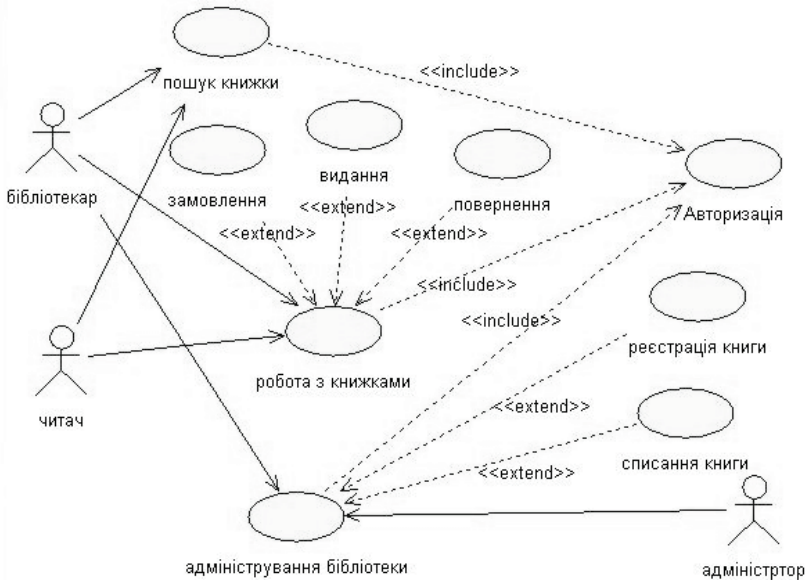


Рисунок 4.1 – Діаграма прецедентів

Проектована система представляється у вигляді множини сутностей чи акторів, взаємодіючих із системою за допомогою варіантів використання (use case), що слугують для опису сервісів, що система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій системи при діалозі з актором.

4.2.2 Діаграма станів

Діаграми станів (рис. 4.2) зображають всі можливі стани, в яких може знаходитися конкретний об'єкт, а також зміни стану об'єкту, які

відбуваються в результаті впливу деяких подій на цей об'єкт. У більшості об'єктно-орієнтованих методів діаграми станів будуються для єдиного класу, щоб показати динаміку поведінки єдиного об'єкту.

Зазвичай діаграми станів складаються з наступних елементів:

- прості та складені стани;
- переходи разом з асоційованими подіями та діями.

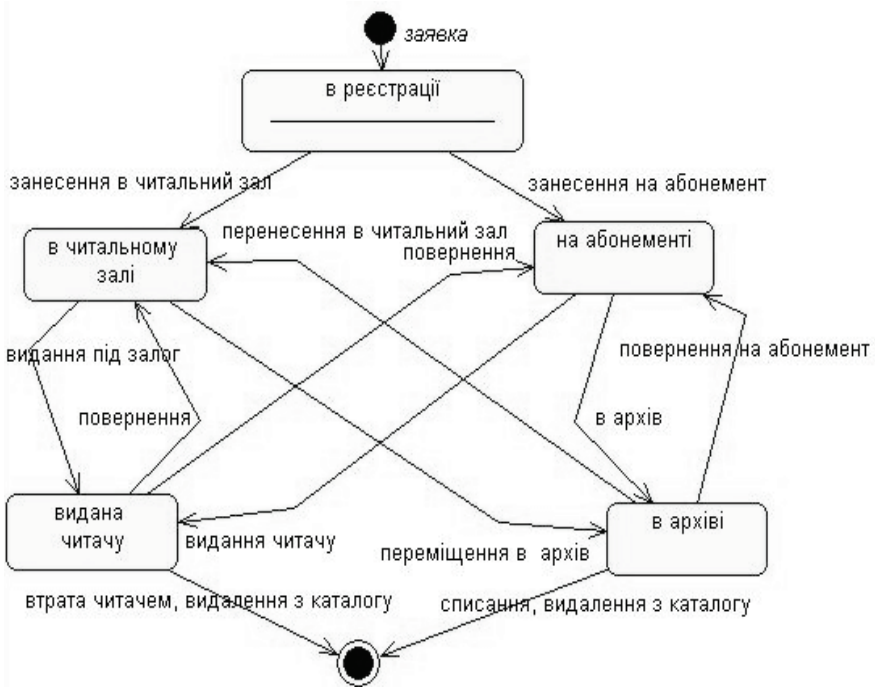


Рисунок 4.2 – Іменовані переходи між станами

4.2.3 Діаграма діяльності

Діаграма діяльності показує переходи між видами діяльності. Модель видів діяльності (activity model) може подавати в графічній формі потік подій для прецеденту (рис. 4.3).

Кожен прецедент можна моделювати за допомогою одного або декількох графів видів діяльності. Діаграма діяльності за своєю сутні-

стю є блок-схемою, яка демонструє, як потік управління переходить від однієї діяльності до іншої.

Подія, джерелом якої служить суб'єкт-прецедент, що ініціює, це та ж сама подія, що запускає виконання графа видів діяльності. Процес виконання послідовно переходить від одного стану виду діяльності до іншого.

Діаграма діяльності у загальному випадку складається зі:

- станів діяльності та станів дії;
- переходів.

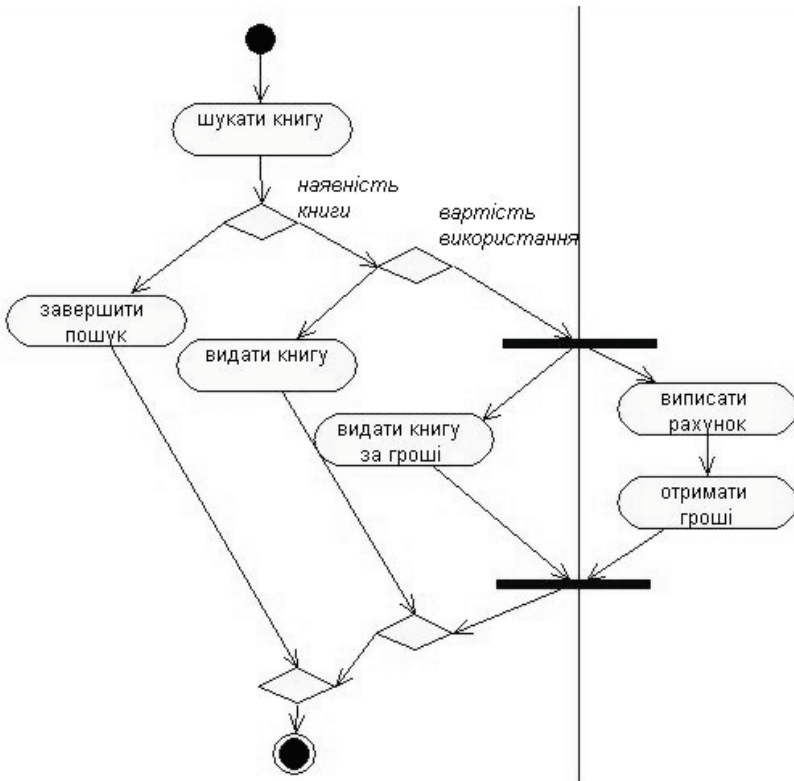


Рисунок 4.3 – Діаграма діяльності

Перехід – це відношення між двома станами, що демонструє те, що об'єкт, який знаходиться в першому стані, повинен виконати деякі дії та перейти у другий стан, як тільки відбудеться вказана подія та будуть задоволені вказані умови.

Розгалуження описує різноманітні шляхи виконання в залежності від значення деякого булівського виразу.

Послідовні переходи в діаграмах діяльності виконуються найчастіше, однак використовуються також і паралельні потоки. У UML для позначення розділення та злиття таких паралельних потоків виконання використовується синхронізаційна лінія, яка зображується у вигляді жирної вертикальної або горизонтальної лінії.

4.3 Завдання на лабораторну роботу

4.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

4.3.2 Вивчити правила побудови діаграм поведінки: діаграм діяльності, станів, прецедентів.

4.3.3 Розробити абстрактну модель системи за обраною темою на основі діаграми прецедентів.

4.3.4 Розробити діаграми прецедентів для представлення варіантів використання системи, наведених на загальній діаграмі прецедентів.

4.3.5 Розробити абстрактну модель системи за обраною темою на основі діаграми діяльності.

4.3.6 Розробити абстрактну модель системи за обраною темою на основі діаграми станів.

4.3.7 Оформити звіт з роботи.

4.3.8 Відповісти на контрольні питання.

4.4 Зміст звіту

4.4.1 Тема та мета роботи.

4.4.2 Тема, обрана для проектування.

4.4.3 Загальна схема роботи системи.

4.4.4 Діаграма діяльності системи, що проектується.

4.4.5 Діаграма станів системи, що проектується.

4.4.6 Діаграми прецедентів системи, що проектується.

4.4.7 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

4.5 Контрольні запитання

4.5.1 Які види діаграм поведінки розрізняють в UML?

4.5.2 Які аспекти поведінки системи моделюються в діаграмах поведінки?

4.5.3 З чого складається діаграма діяльності?

4.5.4 Якими спільними з іншими видами діаграм властивостями володіє діаграма поведінки?

4.5.5 Для чого використовуються діаграми діяльності?

4.5.6 Для моделювання яких об'єктів використовуються діаграми стану?

4.5.7 Яке значення мають терміни “автомат”, “стан”, “подія”, “перехід”, “діяльність”, “дія” в сенсі діаграми стану?

4.5.8 Наведіть приклади застосування діаграм прецедентів.

4.5.9 Які елементи включають в себе діаграми прецедентів?

4.5.10 Який вид діаграм поведінки може бути використано для моделювання операції?

5 ЛАБОРАТОРНА РОБОТА № 5 ПРОЕКТУВАННЯ З ВИКОРИСТАННЯМ UML. ДІАГРАМИ ВЗАЄМОДІЇ

5.1 Мета роботи

Ознайомитися з видами діаграм взаємодії та особливостями їх застосування, навчитися розробляти абстрактну модель системи мовою UML на основі діаграм взаємодії.

5.2 Короткі теоретичні відомості

5.2.1 Діаграма кооперації

Діаграма кооперації (рис. 5.1) призначена для специфікації структурних аспектів взаємодії. Головна особливість діаграми кооперації полягає в можливості графічно уявити не тільки послідовність взаємодії, але і всі структурні відносини між об'єктами, що беруть участь в цій взаємодії.

Для створення діаграм кооперації необхідно розташувати об'єкти, що беруть участь у взаємодії, у вигляді вершин графа. Потім зв'язки, що з'єднують ці об'єкти, відображаються у вигляді дуг даного графу. Зв'язки доповнюються повідомленнями, які об'єкти приймають та посилають.

Діаграма кооперації має дві властивості, які відрізняють її від діаграми послідовності:

а) шлях: для опису зв'язку одного об'єкта з іншим до дальньої кінцевої точки цього зв'язку можна приєднати стереотип шляху (наприклад, `local` показує, що зазначений об'єкт є локальним по відношенню до відправника повідомлення); має сенс явним чином зображати шлях зв'язку тільки по відношенню до шляхів типу `local`, `parameter`, `global` та `self`;

б) порядковий номер повідомлення: для позначення часової послідовності перед повідомленням можна поставити номер, який повинен поступово збільшуватись для кожного нового повідомлення; для позначення вкладеності використовується десяткова нотація Дьюї.

Найчастіше моделюються нерозгалужені послідовності потоків управління. Однак можливо моделювати і більш складні потоки, що містять ітерації та розгалуження.

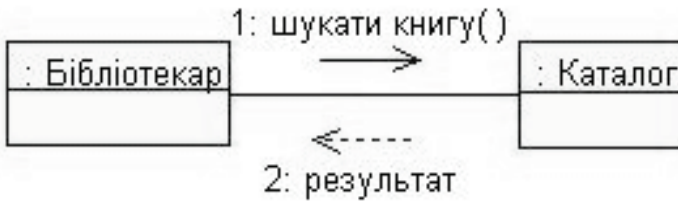


Рисунок 5.1 – Діаграма кооперації

5.2.2 Діаграма послідовності

Діаграма послідовності показує учасників взаємодій і послідовність повідомлень, якими вони обмінюються.

На діаграмі послідовності (рис. 5.2) зображуються виключно ті об'єкти, що безпосередньо беруть участь у взаємодії і не показуються можливі статичні асоціації з іншими об'єктами. Для діаграми послідовності ключовим моментом є саме динаміка взаємодії об'єктів у часі.

Для створення діаграми послідовності потрібно, по-перше, розташувати об'єкти, що беруть участь у взаємодії, у верхній її частині вздовж вісі Х. Зазвичай об'єкт, що ініціює взаємодію, розташовують зліва, а інші – правіше (чим далі, тим більш підпорядкованим є об'єкт). Потім вздовж вісі Y розташовують повідомлення, які об'єкти надсилають та приймають, при чому пізніші розташовуються нижче. Це надає наочну картину, яка дозволяє зрозуміти розвиток потоку керування в часі.

Діаграми послідовності характеризуються двома особливостями, які відрізняють їх від діаграм кооперації.

По-перше, на них є лінія життя об'єкту – вертикальна пунктирна лінія, що відбиває існування об'єкту в часі. Більша частина об'єктів, представлених на діаграмі, існує протягом всієї взаємодії, тому їх зо-

бражують у верхній частині діаграми, а їх лінії життя промальовані згори донизу.

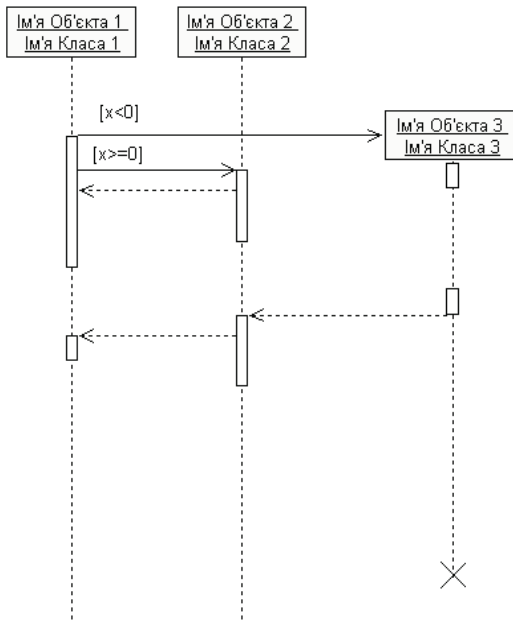


Рисунок 5.2 – Діаграма послідовностей

Об'єкти можуть створюватись і під час взаємодії. Лінія життя таких об'єктів починаються з отримання повідомлення зі стереотипом create. Об'єкти можуть також знищуватись під час взаємодії: у такому випадку їх лінії життя завершуються отриманням повідомлення зі стереотипом destroy, а в якості візуального образу використовується велика літера X, що позначає кінець життя об'єкту.

Друга особливість – фокус керування. Він позначається витягнутим прямокутником, що показує проміжок часу, протягом якого об'єкт виконує деяку дію безпосередньо або за допомогою підпорядкованої процедури. Верхня грань прямокутника вирівнюється за часовою віссю з моментом початку дії, нижня – з моментом його завершення. Вкладеність фокусу керування, викликану рекурсією або зво-

ротним викликом з боку іншого об'єкту, можна продемонструвати, розташувавши інший фокус керування правіше свого батьківського об'єкту.

5.3 Завдання на лабораторну роботу

5.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

5.3.2 Вивчити правила побудови діаграм взаємодії: діаграм кооперації, послідовності, синхронізації, огляду взаємодії.

5.3.3 Розробити абстрактну модель системи за обраною темою на основі діаграм кооперації.

5.3.4 Розробити абстрактну модель системи за обраною темою на основі діаграм послідовності.

5.3.5 Оформити звіт з роботи.

5.3.6 Відповісти на контрольні питання.

5.4 Зміст звіту

5.4.1 Тема та мета роботи.

5.4.2 Тема, обрана для проектування.

5.4.3 Загальна схема роботи системи.

5.4.4 Діаграма кооперації системи, що проектується.

5.4.5 Діаграма послідовності системи, що проектується.

5.4.6 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

5.5 Контрольні запитання

5.5.1 Які види діаграм взаємодії розрізняють в UML?

5.5.2 Які види діаграм в UML використовуються для моделювання динамічних аспектів системи?

5.5.3 За яким фактором впорядковується взаємодія об'єктів на діаграмах послідовності?

5.5.4 Чим відрізняються діаграми кооперації від діаграм послідовності?

5.5.5 Що таке сценарій в термінах діаграм послідовності?

5.5.6 Що таке лінія життя об'єкту і як вона позначається на діаграмі послідовності?

5.5.7 Що таке фокус керування і як він позначається на діаграмі послідовності?

5.5.8 Скільки фокусів керування може мати об'єкт протягом своєї лінії життя?

5.5.9 Хто може ініціювати взаємодію в системі?

5.5.10 Які існують різновиди повідомлень в UML?

6 ЛАБОРАТОРНА РОБОТА № 6 АНАЛІЗ АРХІТЕКТУРИ ВЕБ-ДОДАТКІВ НА ОСНОВІ ПРЕДМЕТНОЇ ОБЛАСТІ

6.1 Мета роботи

Навчитися розробляти та проводити аналіз архітектури веб-додатків на основі предметної області.

6.2 Короткі теоретичні відомості

Для побудови архітектурного каркасу додатку використовують схему проектування Model-view-controller (MVC). Це схема використання декількох шаблонів проектування, за допомогою яких модель даних програми, користувацький інтерфейс і взаємодія з користувачем розділені на три окремих компоненти.

Логіка відображення (front end, frontend) – відповідає за збір даних від користувачів в різних формах та обробку їх у відповідності зі специфікацією.

Логіка управління даними (back-end, backend) – обробляє дані, отримані від користувача для передачі на наступний рівень.

Бізнес-логіка – сукупність правил, принципів, залежностей поведінки об'єктів предметної області. Це реалізація предметної області в інформаційній системі.

Особливості розробки архітектури веб-додатку на основі предметної області:

- можливість більш докладного опису логіки взаємодії об'єктів предметної області;
- можливість гнучкої зміни логіки роботи додатку;
- відсутність надлишкового коду в кінцевій реалізації;
- поділ логіки відображення, логіки управління даними і бізнес-логіки.

Недоліки розробки архітектури веб-додатку на основі предметної області:

- велика трудомісткість при створенні проекту;
- більш жорсткі вимоги до кваліфікації розробника.

6.3 Завдання на лабораторну роботу

6.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

6.3.2 Проаналізувати предметну область.

6.3.3 Розробити архітектуру веб-додатку на основі предметної області.

6.3.4 Оформити звіт з роботи.

6.3.5 Відповісти на контрольні питання.

6.4 Зміст звіту

6.4.1 Тема та мета роботи.

6.4.2 Тема, обрана для проектування.

6.4.3 Загальна схема роботи системи.

6.4.4 Схематичне зображення архітектури веб-додатку на основі предметної області.

6.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

6.5 Контрольні запитання

6.5.1 Які недоліки несе в собі проектування веб-додатків на основі предметної області?

6.5.2 Наведіть узагальнену схему веб-додатку на основі предметної області.

6.5.3 Що таке frontend-частина веб-додатку?

6.5.4 Що таке backend-частина веб-додатку?

6.5.5 Що таке бізнес-логіка?

7 ЛАБОРАТОРНА РОБОТА № 7

АНАЛІЗ АРХІТЕКТУРИ ВЕБ-ДОДАТКІВ З ВИКОРИСТАННЯМ CMS

7.1 Мета роботи

Навчитися розробляти архітектуру веб-додатків з використанням CMS, провести порівняльний аналіз архітектур веб-додатків, розроблених на основі предметної області та за використання CMS.

7.2 Короткі теоретичні відомості

CMS (Content Management System – система управління вмістом) – програмний комплекс, що надає функції створення, редагування, контролю та організації структури веб-додатку (сайту), а також текстової та графічної інформації (вмісту).

Переваги використання CMS при розробці:

- протестований і перевірений багатьма розробниками код;
- універсальність;
- простота використання.

Недоліки використання CMS при розробці:

- обмеження предметної області тими об'єктами, які передбачені в CMS;
- визначення основної логіки роботи програми розробниками CMS. Внесення змін в логіку проблематично;
- зайва універсальність і надлишковий функціонал.

CMS поділяються на ECMS (Enterprise Content Management System – система управління вмістом масштабу підприємства) та WCMS (Web Content Management System – система управління веб-вмістом).

WCMS складається з клієнтської та серверної частин.

Клієнтська частина – веб-сайт, доступний для відвідувачів та зареєстрованих користувачів.

Серверна частина містить шар адміністрування веб-сайту, з яким взаємодіє адміністратор, де виконується конфігурація, обслуговування, очищення, генерація статистики та створення нового контенту.

Wordpress, Joomla та Drupal є прикладами безкоштовних CMS.

WordPress дозволяє створювати сайти різних типів, інформаційні, новинні тощо, але в першу чергу використовується для створення блогів або нескладних сайтів інформаційного типу.

Недоліки WordPress: не досить швидка робота сайту, можливість збоїв у випадку високої відвідуваності.

Joomla має більш широку галузь застосування порівняно з Wordpress і загалом є універсальною. Не позбавлена проблем зі швидкістю роботи за високої відвідуваності.

Drupal може використовуватись для створення форумів, блогів, онлайн-енциклопедій, сайтів спільнот. Проте є менш універсальною порівняно з Joomla. До недоліків відносять слабе використання об'єктних можливостей PHP та відсутність зворотної сумісності API.

У таблиці 7.1 представлено порівняння розглядаємих CMS за їх основними характеристиками.

Таблиця 7.1 – Порівняння CMS Drupal, Joomla та WordPress

Характеристика / CMS	Drupal	Joomla	WordPress
Підтримка шаблонів	Добре та зручно розроблено, використовується модульність	Розроблено не дуже добре, немає шаблонізації окремих елементів	Просто та добре розроблено, використовуються звичайні PHP-функції без мов шаблонів
Розширена функціональність	Розроблено добре, використовується модульність	Розроблено дуже добре, наявна система розширень	Розроблено добре, розширення за рахунок плагінів
Безпека	Розроблено добре, оперативне усунення помилок	Розроблено не дуже добре, проблеми зі зломом та оперативністю	Розроблено не дуже добре, є проблеми з сумісністю
Візуальний редактор	Розроблено добре, гнучке та потужне налаштування	Розроблено добре, є спеціальні редактори	Розроблено добре, є розширений редактор

Продовження таблиці 7.1			
Підтримка сучасних технік	Розроблено дуже добре, швидко вбудовуються плагіни	Розроблено добре, компонентний рівень впровадження	Розроблено добре, працює на рівні “движка”

7.3 Завдання на лабораторну роботу

7.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

7.3.2 Обрати одну з систем керування вмістом для розроблення веб-додатку (CMS): *WordPress, Joomla, Drupal*.

7.3.3 Ознайомитись з архітектурою обраної CMS.

7.3.4 Розробити архітектуру веб-додатку на основі використання обраної CMS.

7.3.5 Порівняти архітектуру веб-додатку, розроблену за використання CMS, з архітектурою, розробленою на основі предметної області. Визначити, який варіант розроблення архітектури є більш ефективним для обраної теми проектування.

7.3.6 Оформити звіт з роботи.

7.3.7 Відповісти на контрольні питання.

7.4 Зміст звіту

7.4.1 Тема та мета роботи.

7.4.2 Тема, обрана для проектування.

7.4.3 Загальна схема роботи системи.

7.4.4 Схематичне зображення архітектури веб-додатку (на основі CMS).

7.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

7.5 Контрольні запитання

7.5.1 Які переваги має проектування веб-додатків на основі CMS порівняно з проектуванням на основі предметної області?

7.5.2 Які недоліки має проектування веб-додатків на основі CMS порівняно з проектуванням на основі предметної області?

7.5.3 Наведіть узагальнену схему веб-додатку на основі CMS.

7.5.4 За якого варіанту побудови архітектури зв'язність між компонентами веб-додатку є вищою?

7.5.5 Які функціональні можливості надають системи керування вмістом?

7.5.6 Які існують типи CMS?

7.5.7 Якими характеристиками повинна володіти сучасна CMS?

7.5.8 Якими перевагами і якими недоліками відзначаються CMS WordPress, Joomla та Drupal?

8 ЛАБОРАТОРНА РОБОТА № 8 ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ ВЕБ-ДОДАТКІВ

8.1 Мета роботи

Навчитися проектувати інтерфейс веб-додатків, враховуючи вимоги до інтерфейсу замовника та ергономічні показники інтерфейсу.

8.2 Короткі теоретичні відомості

Інтерфейс (від англ. Interface – поверхня розділу, перегородка) – сукупність засобів і методів взаємодії між елементами системи. Залежно від контексту, поняття застосовне як до окремого елемента (інтерфейс елемента), так і до зв'язків елементів (інтерфейс сполучення елементів).

Інтерфейс користувача – сукупність засобів, за допомогою яких користувач спілкується з різними пристроями:

інтерфейс командного рядка – інструкції програмою або пристроєм здійснюються шляхом введення з клавіатури текстових рядків;

графічний інтерфейс – управління програмними функціями реалізовано графічними елементами екрану.

Багато кількісних та евристичних методів використовуються для аналізу та вивчення інтерфейсів. Кількісні методи часто можуть зводити спірні питання до простих обчислень. Ще однією, більш важливою, перевагою даних методів є те, що вони допомагають зрозуміти найважливіші аспекти взаємодії людини з машиною.

8.2.1 Модель GOMS

Класична модель GOMS – «правила для цілей, об'єктів, методів і виділення» (the model of goals, objects, methods, and selection rules) – дозволяє передбачити, скільки часу буде потрібно досвідченому користувачеві на виконання конкретної операції при використанні даної моделі інтерфейсу.

GOMS розділяє взаємодію користувачів з комп'ютером на елементарні дії (ці дії можуть бути фізичними, пізнавальними або діями

сприйняття). Інтерфейс досліджується за допомогою цих елементарних дій в якості основи.

Хоча для різних користувачів час виконання того чи іншого жесту може суттєво різнитися, дослідники виявили, що для більшої частини порівняльного аналізу задач, що включають використання клавіатури та графічного пристрою вводу, замість проведення вимірів для кожного окремого користувача можна використати набір стандартних інтервалів. Так, наприклад, в оригінальній номенклатурі на натиснення клавіші відводиться 0.2 с, 1.1 с – час, необхідний користувачу для того, щоб вказати на якусь позицію на екрані монітору, 0.4 с – час, необхідний користувачу для того, щоб перемістити руку з клавіатури на графічний пристрій виводу або навпаки, 1.35 с – час, необхідний користувачу для того, щоб розумово підготуватись до наступного кроку, R – час, протягом якого користувач повинен очікувати відповідь комп'ютера. На практиці вказані значення можуть варіюватися в широких інтервалах. Для досвідченого користувача, який може друкувати зі швидкістю 135 слів/хв., час на натиснення клавіші може становити 0.08 с, для звичайного користувача, що має швидкість 55 слів/хв., – 0.2 с. Окрім того швидкість набору залежить і від того, що саме набирається.

Тим не менш за допомогою типових значень можна виконати правильне порівняльне оцінювання між двома інтерфейсами за рівнем ефективності їх використання.

Тривалість відповіді від комп'ютера може призводити до неочікуваного ефекту на дії користувача. Якщо в процесі використання якогось керуючого елемента на екрані монітору протягом приблизно 250 мс нічого не відбувається, користувач, швидше за все, почне відчувати неспокій, вирішить зробити ще одну спробу або зробить припущення, що система несправна. Якщо затримки неминучі, важливо, щоб в інтерфейсі було передбачений зворотний зв'язок, що повідомляє про них.

Розроблення інтерфейсу починається з визначення задач або набору задач, для яких продукт призначений. Обчислення часу, необхідного для виконання тієї чи іншої дії, за допомогою моделі GOMS починається з перелічення операцій зі списку жестів моделі GOMS, які входять до даної дії.

8.2.2 Закон Фіттса

Закон Фіттса свідчить, що час досягнення мети зворотно-пропорційний розміру цілі та дистанції до неї. Закон Фіттса – загальний закон, що стосується сенсорно-моторних процесів, він зв'язує час руху з точністю руху і з відстанню переміщення: чим далі чи точніше виконується рух, тим більше корекції необхідно для його виконання і, відповідно, більше часу потрібно для внесення цієї корекції.

В інтерфейсі програми довжина прямої лінії, що з'єднує початкову позицію курсора і найближчу точку цільового об'єкта, визначається у законі Фіттса як дистанція. На основі даних про розміри об'єкту і дистанції закон Фіттса дозволяє знайти середній час, за який користувач зможе перемістити курсор до кнопки.

Математично закон записується таким чином:

$$T = a + b \log_2 \left(\frac{D}{W} + 1 \right),$$

де T – середній час, що витрачається на вчинення дії;

a – середній час запуску / зупинки руху;

b – величина, що залежить від типової швидкості руху;

D – дистанція від точки старту до центру цілі;

W – ширина цілі, виміряна уздовж вісі руху.

Для приблизних обчислень можна використати наступні значення констант: $a = 50$, $b = 150$.

8.2.3 Закон Хіка

Закон Хіка говорить про те, що чим менше елементів меню, тим менше часу займає вибір одного з них.

Перед тим як перемістити курсор до мети або вчинити будь-яку іншу дію з набору безлічі варіантів, користувач повинен вибрати цей об'єкт або дію. У законі Хіка стверджується, що коли необхідно зробити вибір з n варіантів, час на вибір одного з них буде пропорційним логарифму за основою 2 від числа варіантів плюс 1 за умови, що всі варіанти є рівно ймовірними. У даному вигляді закон Хіка схожий на закон Фіттса:

$$T = a + b \log_2 (n + 1).$$

Якщо ймовірність першого варіанту дорівнює p_i , то замість логарифмічного коефіцієнту використовується:

$$H = \sum_{i=1}^n p_i \log_2 \left(\frac{1}{p_i} + 1 \right),$$

де p_i означає ймовірність i -го варіанту за умови теоретико-інформаційної ентропії.

Якщо варіанти для вибору представлені незрозумілим чином, значення a і b зростають. Наявність навичок і звичок у користувача при використанні системи знижує значення b .

8.3 Завдання на лабораторну роботу

8.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

8.3.2 Проаналізувати вимоги до інтерфейсу ПЗ на основі документу опису вимог та ТЗ.

8.3.3 Визначити особливості роботи кожної групи користувачів з кожним елементом взаємодії користувача з веб-додатком.

8.3.4 Розробити інтерфейс для веб-додатку, що реалізує систему.

8.3.5 Обґрунтувати вибір запропонованих елементів інтерфейсу, ґрунтуючись на вимогах до інтерфейсу та ергономічних показниках інтерфейсу.

8.3.6 Оформити звіт з роботи.

8.3.7 Відповісти на контрольні питання.

8.4 Зміст звіту

8.4.1 Тема та мета роботи.

8.4.2 Тема, обрана для проектування.

8.4.3 Загальна схема роботи системи.

8.4.4 Інтерфейс веб-додатку за обраною темою.

8.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

8.5 Контрольні запитання

- 8.5.1 Які ергономічні показники інтерфейсу можна виділити?
- 8.5.2 Які існують методи кількісного аналізу інтерфейсу?
- 8.5.3 Що дозволяє передбачити модель GOMS?
- 8.5.4 Які жести розрізняє модель GOMS?
- 8.5.5 Що визначає закон Фіттса?
- 8.5.6 Що визначає закон Хіка?

9 ЛАБОРАТОРНА РОБОТА № 9 АНАЛІЗ СТРУКТУРИ ВЕБ-ДОДАТКІВ

9.1 Мета роботи

Навчитися розробляти структуру веб-додатків.

9.2 Короткі теоретичні відомості

9.2.1 Базові структурні компоненти CMS

База даних. CMS, як правило, забезпечують зберігання даних в єдиному репозиторії (сховищі).

Панель управління – окрема частина програми, що має розширений доступ і дозволяє спростити обслуговування ресурсу.

Ядро системи – основна частина CMS, яка управляє показом сторінок користувачеві, всієї навігацією, відповідно реалізує всі додаткові функції, покладені на нього.

Шаблонізатор – частина системи, що дозволяє розробляти і впроваджувати інтерфейс, не торкаючись ядра системи.

Додаткові модулі – незалежні частини веб-додатку, що дозволяють розширити його функціональність на базі ядра системи.

9.2.2 Класифікація CMS

Класифікації за моделлю будови CMS:

Модульна модель. Такі CMS представляють собою набір окремих суб-CMS, кожна з яких управляє своїм набором типів даних. Модулі незалежні і повністю відповідають за роботу з документами даного типу. Найчастіше такий принцип організації є найвдалішим, оскільки розширювати функціональність можна за рахунок додавання нового модуля, заміни або редагування існуючого коду.

Об'єктна модель. CMS цього типу працюють з класами і об'єктами. Класи визначають структуру даних і набір атрибутів (властивостей). Об'єкт є екземпляром класу, його завдання – зберігати в собі реальні дані. Об'єкт може успадковувати властивості, зміст і поведінку об'єктів, які в них містяться.

CMS також розрізняються за місцем зберігання згенерованих сторінок:

Динамічні. Сторінка формується динамічно, як тільки приходить запит. Це найбільш ресурсномісткий варіант, оскільки CMS працює постійно. Якщо запитів більше певного порогу, то можливе надмірне використання ресурсів сервера. Головна перевага такої схеми – кожен відвідувач отримує найактуальніший варіант вмісту. Крім того, можливе індивідуальне налаштування сторінки під кожного користувача.

З динамічним складанням. Динамічно зібрана (після відповідної команди) сторінка кешується до тих пір, поки для неї не приходить оновлення (чергова команда на регенерацію). В цей момент кеш скидається, і цикл повторюється. Можливо враховувати користувальницькі переваги: сторінка щоразу збирається динамічно, але із статичних (згенерованих заздалегідь) фрагментів. Якщо який-небудь фрагмент оновлюється, то при наступному запиті він (один фрагмент, а не всі взагалі) буде створений заново і занесений в кеш.

Генерація статичних сторінок на основі динамічних даних. При оновленні CMS регенерує всі зв'язані сторінки, тому при запиті видається вже сформована статична сторінка.

9.3 Завдання на лабораторну роботу

9.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

9.3.2 Розробити структуру веб-додатку на основі використання обраної системи керування вмістом CMS (або на основі предметної області, якщо обрано такий варіант побудови архітектури).

9.3.3 Оформити звіт з роботи.

9.3.4 Відповісти на контрольні питання.

9.4 Зміст звіту

9.4.1 Тема та мета роботи.

9.4.2 Тема, обрана для проектування.

9.4.3 Загальна схема роботи системи.

9.4.4 Структура веб-додатку, що реалізує систему.

9.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

9.5 Контрольні запитання

9.5.1 Які компоненти містить веб-додаток?

9.5.2 Чим визначається структура веб-додатків?

9.5.3 Яким чином обрана структура веб-додатку впливає на безпеку?

9.5.4 Які ресурси містить кореневий каталог веб-додатків?

10 ЛАБОРАТОРНА РОБОТА № 10 РОЗГОРТАННЯ ВЕБ-ДОДАТКІВ

10.1 Мета роботи

Навчитися розгортати одну з систем керування вмістом CMS на сервері.

10.2 Короткі теоретичні відомості

Веб-сервер – це сервер, який приймає HTTP-запити від клієнтів, зазвичай веб-браузерів, і видає їм HTTP-відповіді разом із затребуваними даними: HTML-сторінкою, зображенням, файлом, медіа-потокom і т.д.

На серпень 2011 року найбільш поширеним веб-сервером, що займає більше 65% ринку, є вільний веб-сервер Apache.

Ядро Apache включає в себе основні функціональні можливості, такі як обробка конфігураційних файлів, протокол HTTP і система завантаження модулів. Apache HTTP Server підтримує модульність.

У модулях реалізуються такі можливості, як:

- підтримка мов програмування;
- додавання функціоналу;
- виправлення помилок або модифікація основних функцій;
- посилення безпеки.

LAMP – акронім, що позначає набір (комплекс) серверного ПЗ, який широко використовують в Інтернеті. LAMP названий за першими літерами продуктів, що входять до його складу:

- Linux – операційна система Linux;
- Apache – веб-сервер;
- MySQL – СУБД;
- PHP – мова програмування, що використовується для створення веб-додатків (крім PHP можуть матися на увазі інші мови, такі як Perl і Python).

Аналогічно йому існує WAMP:

- Windows – операційна система від компанії Microsoft;
- Apache;
- MySQL;

– PHP.

Хоча спочатку ці програмні продукти не розроблялися спеціально для роботи один з одним, однак така зв'язка стала вельми популярною через свою гнучкість, продуктивність та низьку вартість.

Розгортання CMS в більшості випадків складається з таких етапів:

- створення бази даних в доступній СУБД (MySQL найчастіше);
- завантаження інсталяційного пакета на сервер;
- попереднє налаштування через систему веб-доступу;
- підключення до бази даних;
- встановлення.

При інсталяції необхідно зазначити, що CMS WordPress та Drupal потребують створення нового користувача та нової бази даних в phpMyAdmin. На відміну від вищезазначених CMS, Joomla не потребує попереднього створення бази даних та користувача, ці дії виконуються під час безпосередньої інсталяції CMS. Для WordPress існує ще один етап інсталяції – створення файлу конфігурації wp-config.php.

Останнім етапом інсталяції є заповнення необхідної інформації про веб-додаток та адміністративну частину. Майже такі самі дії потрібні для встановлення CMS Joomla та Drupal, тільки, на відміну від інших, в CMS Joomla останнім пунктом є видалення каталогу INSTALLATION з хосту на локальному чи віртуальному сервері.

Для редагування вмісту веб-додатку необхідно перейти до панелі адміністратора – це, як правило, сторінка, за допомогою якої адміністратор може створювати, редагувати та вилучати інформацію безпосередньо з веб-додатку. Доступ до цієї панелі мають також деякі групи користувачів. У кожній CMS є свій спосіб входження до панелі адміністратора. Так, в CMS Joomla необхідно перейти за посиланням <http://сайт/administrator/>, в CMS Wordpress – <http://сайт/wp-admin/> (wp – стандартний префікс таблиць бази даних, який можна обрати при інсталяції). Авторизація у CMS Drupal відбувається дещо іншим чином і для того, щоб увійти до адміністративної панелі потрібно спочатку авторизуватись на головній сторінці. Сама панель не займає окремої сторінки і представляє собою «верхнє меню», в якому є всі функції повноцінних панелей адміністратора попередніх систем. Зайшовши до форми авторизації в панелі адміністратора, потрібно ввести логін та пароль, який вводився під час інсталяції CMS.

Веб-додатки, розроблені за допомогою систем управління вмістом, складаються зі статей та модулів. У WordPress замість модулів використовуються плагіни. Як правило, сторінки на будь-якій CMS містять в собі верхній та нижній колонтитули, верхнє меню, головне меню та форму авторизації. Головна сторінка веб-додатку розподіляється на декілька колонок, кожна з яких в свою чергу складається з комірок, де і розташовуються модулі та плагіни.

10.3 Завдання на лабораторну роботу

10.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

10.3.2 Виконати конфігурацію сервера та, якщо обрано підхід на основі CMS, інсталивати обрану CMS на сервер відповідно до інструкцій для даної CMS.

10.3.3 Оформити звіт з роботи.

10.3.4 Відповісти на контрольні питання.

10.4 Зміст звіту

10.4.1 Тема та мета роботи.

10.4.2 Тема, обрана для проектування.

10.4.3 Загальна схема роботи системи.

10.4.4 Алгоритм розгортання обраної CMS (серверного програмного забезпечення) на сервері.

10.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

10.5 Контрольні запитання

10.5.1 Що таке розгортання?

10.5.2 Як виконати конфігурацію сервера?

10.5.3 Що таке веб-сервер?

10.5.4 Що таке LAMP та WAMP?

10.5.5 Чим відрізняється інсталяція CMS WordPress, Joomla та Drupal?

11 ЛАБОРАТОРНА РОБОТА № 11 АНАЛІЗ ЯКОСТІ ТА ОЦІНКА ВЕБ-ДОДАТКІВ

11.1 Мета роботи

Навчитися визначати якість веб-додатків на основі проведення тестування.

11.2 Короткі теоретичні відомості

Тестування ПЗ (Software Testing) – це процес технічного дослідження, який виконується на вимогу замовників, і призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись.

Якість ПЗ – характеристика ПЗ, ступінь відповідності ПЗ вимогам. Частіше за все, використовують визначення ISO 9001, згідно з яким якість – це «ступінь відповідності наявних характеристик вимогам».

Фактори якості – це нефункціональні вимоги до ПЗ, що відносяться до, наприклад, надійності та продуктивності програм.

Деякі з факторів якості:

Зрозумілість. Призначення ПЗ повинно бути зрозумілим з самої програми та документації.

Повнота. Всі необхідні частини програми повинні бути представлені та реалізовані.

Стислість. Відсутність надлишкової інформації та такої, що дублюється. Реалізація принципів DRY.

Можливість портування. Легкість в адаптації програми до інших умов: архітектури, платформи, операційній системі тощо.

Узгодженість. Вся документація та код повинні виконуватися за єдиними угодами, використовувати єдині формати та позначення

Покриття тестуванням.

Зручність використання.

Надійність.

Безпечність.

За об'єктом тестування розрізняють функціональне тестування, тестування продуктивності (навантажувальне тестування, стрес-

тестування та тестування стабільності), юзабіліті-тестування, тестування інтерфейсу користувача, тестування безпеки, локалізації та сумісності.

Тестування продуктивності – тестування, яке відбувається з метою визначення того, як швидко працює система або її частина під деяким рівнем навантаження.

Навантажувальне тестування – найпростіша форма тестування продуктивності, яке проводиться з метою оцінювання додатку під заданим очікуваним навантаженням. Таким навантаженням зокрема може слугувати очікувана кількість користувачів, що одночасно працюють із застосуванням та виконують задану кількість транзакцій за інтервал часу. Даний тип тестування дозволяє отримати час відгуку найважливіших бізнес-транзакцій.

Стрес-тестування зазвичай використовується для визначення меж пропускнуєї здатності застосування.

Тестування стабільності виконується з метою перевірки того, чи витримає додаток очікуване навантаження впродовж тривалого періоду часу. За такого тестування сама тривалість його проведення не має першочергове значення, основна задача – спостерігаючи за використанням ресурсів, виявити втрати пам'яті та провести спостереження за тим, щоб швидкість обробки даних та/або час відклику застосування на початку тесту та з тривалістю часу не зменшувався, оскільки у протилежному випадку можливі збої в роботі системи.

Юзабіліті-тестування (перевірка ергономічності) – метод оцінювання зручності продукту у використанні, що ґрунтується на залученні користувачів у якості тестувальників. У процесі даного виду тестування визначається група людей, які є потенційними користувачами, яким пропонується виконати в системі ряд завдань, що відповідають ключовим сценаріям використання продукту.

За типом інформації, збір якої виконується, розрізняють 2 види юзабіліті-тестування: якісне (направлене на збір думок користувачів, інформації про найбільш розповсюджені проблеми та причини виникнення) та кількісне (замірюються показники, які є індикаторами продуктивності, результативності та задоволеності продуктом: отримані дані порівнюють з аналогічними характеристиками конкуруючих систем, попередніми версіями, іншими варіантами інтерфейсу або прийнятими на початку розробки цільовими показниками).

Тестування безпеки – оцінювання вразливості ПЗ до різноманітних атак. У процесі тестування безпеки випробувач грає роль зломника і повинен спробувати: дізнатися пароль за допомогою зовнішніх засобів, виконати атаку системи за допомогою спеціальних утиліт, що аналізують захист, ціленаправлене введення помилок в надії проникнути в систему в процесі відновлення, перегляд несекретних даних з метою знайти ключ до входу в систему, заглушення роботи системи.

При цьому необхідно зазначити, що задача проектувальника системи полягає в тому, щоб зробити ціну проникнення більш високою, ніж ціна інформації, що може бути отримана в результаті.

Тестування сумісності – вид нефункціонального тестування, основною метою якого є перевірка коректної роботи продукту у деякому оточенні, параметри якого включають: апаратну платформу, мережеві пристрої, операційна система, база даних, системне ПЗ, браузері, периферія.

За ступенем автоматизації розрізняють ручне, автоматизоване та напівавтоматизоване тестування.

За ступенем ізолюваності компонентів розрізняють модульне, інтеграційне та системне тестування.

Модульне тестування тестує мінімальний компонент програми, або модуля. Кожний модуль тестується для перевірки правильності його реалізації.

Інтеграційне тестування виявляє дефекти в інтерфейсах та у взаємодії між компонентами (модулями).

Системне тестування тестує інтегровану систему для перевірки відповідності всім вимогам.

Системне інтеграційне тестування перевіряє, чи система інтегрується в будь-яку зовнішню систему (або системи) відповідно до системних вимог.

Приймальне тестування може проводитись кінцевим користувачем, замовником, або клієнтом для перевірки, чи може продукт бути прийнятий до використання. **За часом проведення тестування** розрізняють:

- альфа-тестування – це симульоване або реальне операційне тестування потенційними користувачами/замовником або командою тестувальників на боці розробника;

- бета-тестування виконується після альфа-тестування. Версії ПЗ, відомі як бета-версії, надаються у користування обмеженій кіль-

кості людей поза компанією для того, щоб упевнитись, що програма не містить великої кількості помилок.

У термінології професіоналів з тестування терміни тестування “білої скрині” та тестування “чорної скрині” належать до того, чи має розробник тестів доступ до коду ПЗ, що тестується, або ж тестування виконується через користувацький інтерфейс або прикладний програмний інтерфейс, що надається модулем, який тестується.

За тестування “білої скрині” розробник тесту має доступ до коду та може писати код, який пов’язаний з бібліотеками ПЗ, що тестується. За тестування “чорної скрині” тестувальник має доступ до ПЗ тільки через той самий інтерфейс, що й замовник або користувач, або через зовнішні інтерфейси, що надають можливість іншому комп’ютеру або іншому процесу підключитися до системи для тестування. Наприклад, модуль, який виконує тестування, може віртуально натискати клавіші або кнопки миші в тестуємій програмі за допомогою механізму взаємодії процесів із впевненістю в тому, що ці події викликають той самий відгук, що й реальні натиснення клавіш та кнопок миші.

Оскільки основу тестування та управління змінами складає простежуваність, межі та глибину простежуваності проекту потрібно визначати з використанням аналізу витрат та результатів. Щонайменше необхідно простежуваність підтримувати між вимогами на основі прецедентів та дефектами. У більш розвиненій моделі між вимогами-прецедентами та дефектами до маршруту простежуваності можна додати тестові вимоги. У ще більш складній моделі простежуваності межі простежуваності можуть включати системні функції, тестові прецеденти, удосконалення, точки тестової верифікації та інші артефакти розроблення ПЗ.

11.3 Завдання на лабораторну роботу

11.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

11.3.2 Розробити тести для аналізу якості веб-додатків.

11.3.3 Провести тестування сумісності.

11.3.4 Проаналізувати реалізованість функціональних вимог до ПЗ.

11.3.5 Перевірити ергономічність веб-додатку.

11.3.6 Провести тестування безпеки веб-додатку.

11.3.7 Провести тестування стабільності.

11.3.8 Оформити звіт з роботи.

11.3.9 Відповісти на контрольні питання.

11.4 Зміст звіту

11.4.1 Тема та мета роботи.

11.4.2 Тема, обрана для проектування.

11.4.3 Загальна схема роботи системи.

11.4.4 Опис тестів, застосованих для аналізу якості веб-додатків, та результатів тестування.

11.4.5 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

11.5 Контрольні запитання

11.5.1 Як можна класифікувати види тестування?

11.5.2 Що таке альфа- та бета-тестування?

11.5.3 Що таке статичне та динамічне тестування?

11.5.4 Які розрізняють фактори якості ПЗ?

11.5.5 Що таке підхід до тестування Capture & Playback?

11.5.6 У чому сутність підходу KeywordDriven до тестування?

11.5.7 Як відбувається тестування навантажувальної здатності?

11.5.8 Яким чином може проводитися перевірка ергономічності?

11.5.9 Чим відрізняється тестування “білої скрині” від тестування “чорної скрині”?

12 ЛАБОРАТОРНА РОБОТА № 12 РОЗШИРЕННЯ ФУНКЦІОНАЛЬНОСТІ ВЕБ- ДОДАТКІВ ЗА ДОПОМОГОЮ МОДУЛІВ

12.1 Мета роботи

Навчитися використовувати модулі розширення функціональності для розроблення веб-додатку.

12.2 Короткі теоретичні відомості

12.2.1 Модульний принцип організації CMS

Більшість CMS зроблені за принципом модульності: є базова основа (ядро), до якої можливе підключення модулів, які надають додаткові функції, яких немає в ядрі CMS. Модулі можуть вже входити до складу комплексу, можуть бути встановлені окремо, а також багато систем допускають самостійну розробку та включення додаткових модулів. Таким чином, загальний функціонал сайту залежить від функцій включених в нього модулів.

Модуль – окреме невелике багатофункціональне рішення для веб-додатку, яке дозволяє вивести яку-небудь інформацію (погоду, лічильники, контент, слайдшоу тощо). Типовими компонентами останніх років є магазин, галерея, розсилка новин або система форумів.

Фактично зміст тієї чи іншої сторінки залежить від модуля, який реалізує функціональний рівень (бізнес-логіку).

Переваги модульних CMS наведено нижче.

Простота сприйняття

Чітко позначені кордони, зв'язність модулів робить їх легкими для розуміння. Вивчення системи невеликими частинами в кінцевому підсумку призводить до більш глибокого її розуміння.

Змінність

Якщо кожен модуль системи відомий тільки через інтерфейс, це дозволяє легко замінити один модуль на інший, з таким само інтерфейсом.

Паралельна розробка

Завдяки своїй незалежності, модулі можуть розроблятися паралельно. Для команди розробників це дає можливість розподілу завдань за межами модулів.

Багаторазовість і гнучкість використання

Кожен модуль, взятий з однієї системи, може бути використаний в іншій. Тут все залежить від його функціоналу і області застосування.

Покращення тестування

Поряд з інтеграційним і модульним тестуванням існує можливість тестування кожного модуля окремо, як єдиного цілого.

12.2.2 Приклади розширення функціональності CMS

Наведемо декілька прикладів модулів, які можна використати для розширення функціональності CMS Drupal, завантаживши з репозиторію на drupal.org:

- Captcha – захисний механізм картинок «CAPTCHA», що використовується при реєстрації;
- Ecommerce, Ubercart – системи електронної комерції;
- FCKeditor, CKEditor, TinyMce – візуальні редактори;
- Gallery – інтеграція з галереєю зображень Gallery2;
- Project – ведення проєктів, включає багтрекер і інтеграцію з CVS і Subversion;
- SPAM – блокування спаму;
- WebForm – гнучкий модуль для швидкого проектування інтерактивних форм (опитування, зворотній зв'язок).

Для інсталяції модуля в Drupal необхідно виконати наступні кроки:

- а) завантажити модуль зі сторінки [drupal.org/project/\[модуль\]](http://drupal.org/project/[модуль]), впевнившись, що версія модуля відповідає версії системи;
- б) розпакувати файли з архіву;
- в) ознайомитись з файлами *README.txt* та *INSTALL.txt*, які містять інформацію про можливості модуля, його інсталяцію та налаштування;
- г) скопіювати папку модуля в системну папку додаткових модулів *sites/all/modules*;
- д) перейти на сторінку «Управління → Створення сайту → Модулі» та включити встановлений модуль;

- е) виконати для модуля необхідне налаштування;
- ж) перейти на сторінку «Управління → Користувачі → Права доступу» та визначити права доступу до модуля.

Модуль CMS Drupal складається з двох файлів, які мають міститись у /sites/all/modules: ім'ямодуля.info та ім'ямодуля.module. Другий файл містить код модуля, а перший – інформацію про модуль: ім'я, опис, версію ядра, які підтримує модуль, а також список модулів, необхідних для роботи даного модуля, та пакет, до якого відноситься модуль.

Приклади плагінів, які використовуються для розширення функціональності ядра CMS Wordpress:

- WordPress Database Backup – дає змогу зробити бекап бази даних;
- MobilePress – створює мобільну версію сайту;
- All in One SEO Pack – дозволяє виконувати пошукову оптимізацію;
- Akismet – допомагає боротися зі спам-коментарями;
- Simple Tags – дозволяє працювати з тегами;
- Related Posts – дозволяє виводити наприкінці кожної публікації список схожих статей.

Для інсталяції плагіну в Wordpress необхідно: завантажити файли обраного плагіну і завантажити їх у папку /public_html/wp-content/plugins/ Wordpress. Після цього можливість активації плагіну з'явиться у адміністративній частині (пункт меню «Модулі»). Натисніть на кнопку «Активувати» і увімкніть плагін.

Для розширення функціональності CMS Joomla можна зокрема використати наступні модулі:

- mod_syndicate – відображає посилання на RSS-стрічку поточної сторінки;
- mod_wrapper – створює у вказаній позиції вікно, в якому відображається сторінка;
- mod_poll – виводить в обрану позицію голосування;
- mod_latestnews – виводить список останніх опублікованих матеріалів;
- mod_related_items – порівнює ключові слова у поточному матеріалі та знаходить у базі даних матеріали з ключовими словами, що співпадають, після чого виводить список схожих матеріалів;
- mod_stats – відображає статистику сайту.

Плагін Wordpress складається з файлу (або файлів) PHP, а також може додатково містити файли JavaScript, CSS, зображення тощо.

На початку файлу PHP необхідно розмістити стандартний інформаційний заголовок (назва плагіну, адреса веб-сторінки з описом плагіну, короткий опис плагіну, номер версії плагіну, ім'я автора та його веб-сторінка) та інформацію про ліцензію.

Для інсталяції модуля в Joomla необхідно виконати наступні кроки:

- а) завантажити модуль;
- б) зайти в адміністративний центр та перейти в “Розширення” => “Менеджер модулів”;
- в) натиснути на кнопку “Створити” на панелі інструментів та обрати необхідний модуль;
- г) налаштувати параметри модуля та натиснути “Зберегти”.

Модуль CMS Joomla складається мінімально з двох файлів: `mod_назвамодуля.xml` та `mod_назвамодуля.php`, які за замовчуванням розташовуються в папці «modules» на сервері.

Файл `mod_назвамодуля.xml` – багатофункціональний файл, який містить: загальні відомості про модуль – назву, опис, авторство, версію; відомості для інсталювання/деінсталювання модуля – список файлів та шляхи їх розміщення; список параметрів для конфігурації модуля.

Файл `mod_назвамодуля.php` містить програмний код модуля.

12.3 Завдання на лабораторну роботу

12.3.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

12.3.2 Ознайомитись з існуючими модулями розширення функціональності для обраної системи керування вмістом.

12.3.3 Проаналізувати вимоги до функціональності, що визначені в оформленій документації на ПЗ.

12.3.4 Визначити необхідні модулі розширення функціональності.

12.3.5 Застосувати обрані модулі для веб-додатку.

12.3.6 Оформити звіт з роботи.

12.3.7 Відповісти на контрольні питання.

12.4 Зміст звіту

12.4.1 Тема та мета роботи.

12.4.2 Тема, обрана для проектування.

12.4.3 Загальна схема роботи системи.

12.4.4 Вимоги до функціональності.

12.4.5 Опис використаних модулів.

12.4.6 Архітектура веб-додатку після використання модулів.

12.4.7 Структура веб-додатку після використання модулів.

12.4.8 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

12.5 Контрольні запитання

12.5.1 Що таке модуль?

12.5.2 Які модулі включає стандартний набір модулів обраної CMS?

12.5.3 З яких файлів складається модуль розширення функціональності?

12.5.4 Як інсталиувати модуль розширення функціональності в обрану CMS?

ЛІТЕРАТУРА

1. Мацяшек, Л.А. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML : Пер. с англ. [Текст] / Л.А. Мацяшек. – М. : Издательский дом “Вильямс”, 2002. – 432 с., ил.
2. Вигерс, К.И. Разработка требований к программному обеспечению : Пер. с англ. [Текст] / К.И. Вигерс. – М. : Издательско-торговый дом "Русская Редакция", 2004. – 576с. : ил.
3. Буч, Г. Язык UML. Руководство пользователя [Текст] / Гради Буч, Джеймс Рамбо, Ивар Якобсон. – М. : ДМК Пресс, 2006. – 496 с.
4. Леоненков, А.В. Самоучитель UML. 2-е издание [Текст] / А.В. Леоненков. – СПб. : "БХВ – Петербург", 2004. – 432 с.
5. Раскин, Дж. Интерфейс : новые направления в проектировании компьютерных систем : Пер. с англ. [Текст] / Джеф Раскин. – СПб. : Символ-Плюс, 2004. – 272 с.
6. Скотт, Б. Проектирование веб-интерфейсов : Пер. с англ. [Текст] / Б. Скотт, Т. Нейл. – СПб. : Символ-Плюс, 2010. – 352 с., ил.
7. Томлинсон, Т. CMS Drupal 7 : руководство по разработке системы управления веб-сайтом, 3-е издание/ Т. Томлинсон. – М. : “Вильямс”, 2011. – 560 с.
8. Граф, Х. Создание веб-сайтов с помощью Joomla! 1.5 / Х. Граф. – М. : “Вильямс”, 2009. – 304 с.