МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ ТА ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

В.В. Семко, О.В.Семко

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ДИСЦИПЛІНИ

ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ТЕСТУВАННЯ

Київ 2023

Розробники:

професор кафедри комп’ютерних наук , д.т.н. Семко В.В.,

старший науковий співробітник ІТГІП НАН України, к.т.н. Семко О.В.

Рецензенти:

доцент кафедри комп’ютерних наук, к. т. н., доцент Голуб Бела Львівна

доцент кафедри комп’ютерних наук, к. ф-м н., Британ Андрый Васильович

Дані методичні вказівки містять завдання та теоретичні відомості для виконання лабораторних робіт з дисципліни «Якість програмного забезпечення та тестування» зі спеціальності 121 «Інженерія програмного забезпечення». Лабораторні роботи спрямовані на придбання студентами професійного рівня підготовки, що дозволяє ефективно вирішувати складні інженерні завдання забезпечення якості автоматизованих систем на усіх етапах життєвого циклу у відповідності з стандартами технологічних процесів.

Інструментальною базою проведення системних досліджень є сучасні технології забезпечення якості виробництва і експлуатації автоматизованих систем та їх компонент.

В методичних вказівках до виконання лабораторних робіт розглядаються питання дослідження сутності технології виробництва та супроводження автоматизованих систем і їх компонент у відповідності до стандартів забезпечення якості на усіх етапах життєвого циклу інформаційних технологій.

Розглянуто на засіданні
кафедри комп'ютерних наук
Протокол №6 від 12.12.2023р.
Схвалено

Розглянуто на вченій раді
факультету інформаційних
технологій
Протокол № від 18.12.2023 р.
Схвалено

ЗМІСТ

Вступ

Сучасні інформаційні та комунікаційні технології відкривають перед виробниками великі потенційні можливості для вирішення фундаментальних завдань ефективного управління часом та вартістю життєвого циклу продукції. При цьому характерними особливостями функціонування виробничих систем при створенні складної наукомної та конкурентоспроможної продукції є висока динамічність і невизначеність середовища, наявність факторів ризику. У зв'язку з цим виникає актуальна потреба у розробці та впровадженні ефективних організаційних технологій забезпечення якості бізнес-процесів з точки зору досягнення бажаного рівня значень основних критеріїв економічної ефективності та підвищення конкурентоспроможності продукції, що виробляється.

Впровадження нових інформаційних та комунікаційних технологій у виробничі системи призводить до необхідності пошуку чи формування ефективних методів та структур, що дозволяють забезпечувати якість технологічного процесу виробництва і експлуатації автоматизованих систем на усіх етапах життєвого циклу у відповідності до діючих стандартів якості..

У переважній більшості інфраструктурна основа сучасних підприємств формується навколо життєвих циклів продукції на основі виробничих інформаційних технологій (розвиток парадигми комп'ютерно-інтегрованих виробництв), а також ділових процесів у руслі концепцій реінжинірингу. При цьому в автоматизації процесів життєвого циклу автоматизованих систем виділяються дві основні групи завдань: управління ресурсами і автоматизація етапів життєвого циклу

В даний час для автоматизованих систем існують моделі життєвого циклу і якості на основі стандартів, які орієнтовані на різні види програмного забезпечення і типи проєктів автоматизованих систем. Ці стандарти базуються на трьох групах процесів: основних (придбання, постачання, розробка та супровід); допоміжних (документування, керування конфігурацією, забезпечення якості, верифікація, атестація, оцінка, аудит); організаційних (визначення, оцінка та вдосконалення життєвішого циклу, стандартизація процесів створення, виробництва, а також тестування, верифікації і сертифікації інформаційних, інформаційно-комунікаційних і програмних систем).

Ключовою метою навчальної дисципліни «Якість програмного забезпечення та тестування» є надання випускникам знань і початкового досвіду, що є необхідним для початку професійної інженерної діяльності. При цьому важливим керівним принципом є включення практичного досвіду

професійної діяльності під час навчання студентів, що спеціалізуються в галузі програмної інженерії.

Знання та навички, засвоєнні під час вивчення цієї дисципліни, студент може використовувати як у подальшому навчанні, так і у своїй професійній діяльності.

Варіанти завдань до лабораторних робіт

1. Програмна система обліку розрахунків з покупцями.
2. Програмна система для проведення соціологічних опитувань.
3. Програмна система продажу квитків в кінотеатрі.
4. Програмна система диспансерного обліку хворих.
5. Програмна система управління ресурсами компанії, що займається виробництвом меблів.
6. Програмна система управління відносинами з постачальниками.
7. Програмна система «Готель».
8. Програмна система обліку оплати послуг інтернет-провайдера.
9. Програмна система обліку канцтоварів на підприємстві.
10. Програмна система управління процесом обробки кореспонденції для компанії експрес доставки поштових відправлень.
11. Програмна система взаємодії з клієнтами туристичної компанії.
12. Програмна система «Біржа праці».
13. Програмна система «Диспетчер таксі».
14. Програмна система «Магазин».
15. Програмна система «Салон краси».
16. Програмна система «Бібліотека».
17. Програмна система «Автосалон».
18. Програмна система обліку розрахунків із ЖЕК.
19. Програмна система «Агенція нерухомості».
20. Програмна система «Автовокзал».
21. Програмна система «Деканат».
22. Програмна система «Страхова компанія».
23. Програмна система «Кафедра».
24. Програмна система «Офіціант».
25. Програмна система продажу квитків в залізничних касах.

Лабораторна робота №1

«Дослідження основних принципів опису процесів і визначення вимог до автоматизованих систем»

Мета роботи: Засвоєння базових знань щодо основних положень щодо аналізу предметної області та вже існуючих систем; ознайомлення з стандартами і дослідження підходів до розробки вимог до створюваних автоматизованих систем.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Ознайомитись з ГОСТ 34.201-89, ГОСТ 19.201-78, ГОСТ 2.102-68, ДСТУ 3973-2000, ГОСТ 2.103-2013, ГОСТ 34.602-89.
4. Дослідити основні принципи опису процесів і визначення вимог до автоматизованих систем.
5. Виконати індивідуальне завдання до лабораторної роботи.
6. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

1 Назва та мета роботи.
2. Коротко теоретичні відомості
3. Завдання дослідження лабораторної роботи згідно до обраного варіанта.
4. Результати виконаного дослідження.
5. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Приклад виконання лабораторної роботи з покроковим описом завдань

Завдання 1. Коротка характеристика програмної системи, яка розробляється, та визначення області застосування.

Програмна система інтернет-магазин продажу програмного забезпечення - це програмна система, яка дозволяє вести детальні записи щодо постачання та продажу програмного забезпечення.

Система інтернет-магазину продажу програмного забезпечення може використовуватися як приватними підприємцями, так і любим користувачем ПК з метою ведення відповідних записів стосовно програмного забезпечення.

Завдання 2. Опис предметної області із приведеною характеристикою бізнес-процесів

Для належної організації роботи системи інтернет-магазину продажу програмного забезпечення необхідне виконання таких основних бізнес-процесів:

- процесу пошуку програмного забезпечення;
- процесу перегляду даних про програмне забезпечення;
- процесу оплати ПЗ;
- процесу отримання ПЗ.

Рис. 1. Діаграма бізнес-процесів розроблюваного програмного продукту

Розглянемо детальніше кожен з вище представлених бізнес-процесів. На рисунку 2 зображено діаграму функцій процесу пошуку програмного забезпечення.

Рис. 2. Діаграма функцій процесу пошуку програмного забезпечення

Першим етапом роботи з будь-яким інтернет-сервісом продажу програмного забезпечення є пошук програмного забезпечення. Цей пошук можна здійснити декількома шляхами, зокрема:

- ❖ пошук типу (є два можливі варіанти: операційні системи і офісні пакети);
- ❖ пошук бренду (ввівши брендпрограмного забезпечення розгорнуться можливі варіанти з вибраним брендом).
- ❖ пошук ціни (ввівши ціну програмного забезпечення розгорнуться можливі варіанти з вибраною ціною).

Характеристику бізнес-процесу пошуку программного забезпечення наведено в таблиці 1.

Таблиця 1 Характеристика бізнес-процесу пошуку програмного забезпечення

Назва характеристики	**Значення характеристики**
Ім'я бізнес-процесу	Пошук програмного забезпечення
Основні учасники	Клієнт
Вхідна подія	Запит на вибір програмного забезпечення
Вхідні документи	Каталог ПЗ
Вихідна подія	Обране ПЗ та його інформація
Вихідні документи	Картка товару
Клієнт бізнес-процесу	Накладна

На рисунку 3 представлена діаграма функцій процесу вибору даних програмного забезпечення, яка складається з двох основних функцій, а саме обрання варіанту програмного забезпечення з таблиці можливих варіантів та

додавання у кошик. Характеристику бізнес-процесу вибору даних програмного забезпечення наведено в таблиці 2.

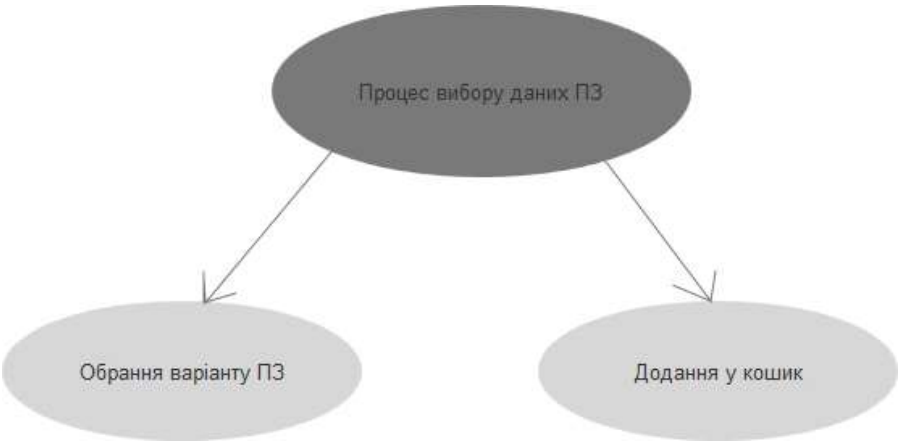


Рис. 3. Діаграма функцій процесу вибору даних програмного забезпечення

Таблиця 2 Характеристика бізнес-процесу вибору даних програмного забезпечення

Назва характеристики	**Значення характеристики**
Ім'я бізнес-процесу	Вибір даних
Основні учасники	Клієнт, система
Вхідна подія	Обраний варіант ПЗ
Вхідні документи	-
Вихідна подія	Додання у кошик ПЗ
Вихідні документи	-
Клієнт бізнес-процесу	Оформлення замовлення

Після етапу пошуку програмного забезпечення та перегляду даних про програмне забезпечення клієнт системи переходить до етапу оформлення замовлення. На рисунку 4 наведені основні функції процесу оформлення ПЗ.

З рисунку видно, що основним функціоналом етапу оформлення замовлення є внесення інформації про покупця (його імені, прізвища та номеру телефону), обрання ПЗ серед тих, що є в наявності, а також обрання форми оплати покупки через наявні он-лайн сервіси оплати. Характеристику бізнес- процесу оформлення замовлення наведено в таблиці 3.

Рис. 4. Діаграма функцій процесу оформлення замовлення

Таблиця 3. Характеристика бізнес-процесу оформлення замовлення

Назва характеристики	**Значення характеристики**
Ім'я бізнес-процесу	Оформлення замовлення
Основні учасники	Клієнт, система
Вхідна подія	Обраний варіант ПЗ, обрана форма оплати, внесені особисті дані
Вхідні документи	Номер картки оплати покупки
Вихідна подія	Оформлене замовлення
Вихідні документи	Накладна
Клієнт бізнес-процесу	Оплата ПЗ

Наступним етапом є оплата програмного забезпечення. На рисунку 5 наведені основні функції процесу оплати ПЗ.

Рис. 5. Діаграма функцій процесу оплати програмного забезпечення

З рисунку видно, що основним функціоналом етапу оплати програмного забезпечення є вибір способу оплати замовлення, введення користувачем платіжних даних та надсилання чеку про оплату.

Таблиця 4. Характеристика бізнес-процесу доставки ПЗ

Назва характеристики	**Значення характеристики**
Ім'я бізнес-процесу	Оплата ПЗ
Основні учасники	Клієнт, система
Вхідна подія	Вибраний спосіб оплати, введені платіжні дані і надсилання чеку
Вхідні документи	Чек
Вихідна подія	Оплачене ПЗ
Вихідні документи	Чек
Клієнт бізнес-процесу	-

3. Огляд і аналіз існуючих аналогів та, на основі аналізу, формування вимог, відповідно, до свого проекту

Для подальшого дослідження було обрано найбільш відомі та функціональні системи:

Софтлист – система, яка призначена для продажу ПЗ, легкої комунікації з покупцями через віконце справа екрану. Вигляд системи зображено на рисунку 6.

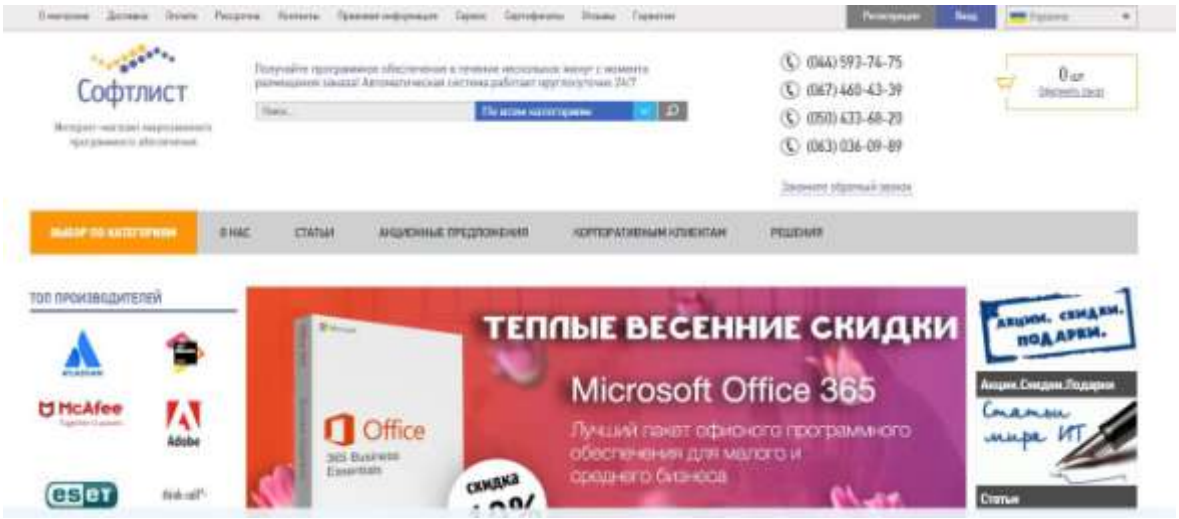


Рис. 6. Головний вигляд сайту **Софтлист**

COMFY – система з зручним дизайном, яка спеціалізується на продажі ПЗ виробника Microsoft.Також є віконце комунікації менеджера з клієнтом. Вигляд системи зображено на рисунку 7.

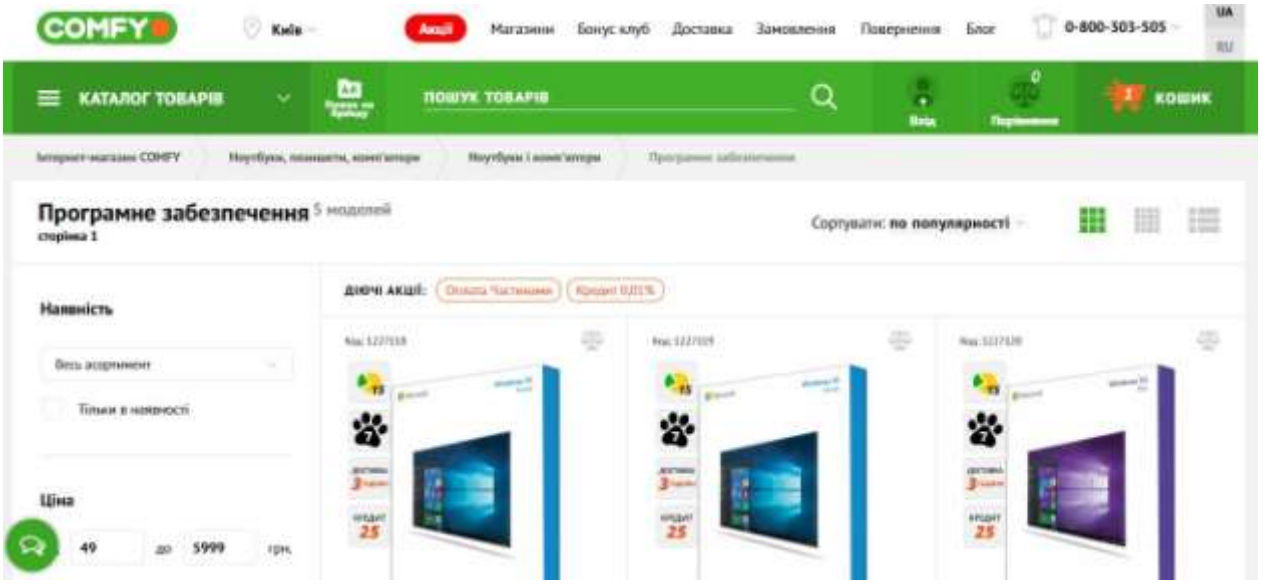


Рис.7. Головний вигляд сайту **COMFY**

АЛЛО – це система продажу ПЗ з великим асортиментом ПЗ та різними варіантами вибору виробника. Вигляд системи зображено на рисунку *8*.

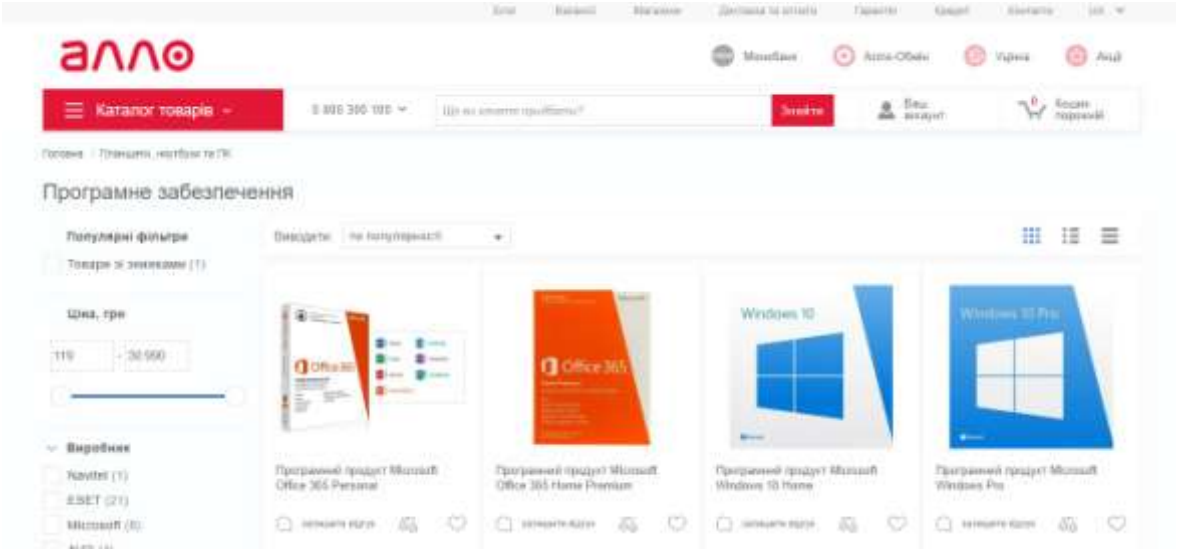


Рис. 8. Головний вигляд сайту **АЛЛО**

Наведені вище системи відрізняються тим, що, зокрема, система «АЛЛО» має більший вибір товару та виробників ніж система «COMFY», яка має лише одного виробника.

Після проведення порівняння цих систем та їх аналізу було виявлено основні їх функціональні і нефункціональні вимоги.

Система	**Вимоги**	
Софтлист	- можливість додавання фотографій товару; - можливість вивантажити товари на сайт магазину; - можливість скласти заявку на замовлення товару.	- можливість паралельної роботи необмеженої кількості користувачів по мережі; - заборона на перегляд собівартості продавцями; - підтримка смартфонів та планшетів; - можливість перегляду товару; - можливість покупки в розстрочку; - можливість комунікації менеджера з клієнтами.
COMFY	- можливість вибору та додавання товару на сайт; - можливість скласти заявку на замовлення товару; - можливість зарезервувати товар; - можливість перегляду списку замовлень; - можливість перегляду товару;	- підтримка смартфонів та планшетів; - максимально простий та приємний інтерфейс; - великі обсяги даних обробляються дуже швидко; - можливість комунікації менеджера з клієнтами.

	- можливість покупки в розстрочку.	
АЛЛО	- можливість прикріплення необмеженої кількості файлів до опису товару; - можливість вибору та додавання товару на сайт; - можливість скласти заявку на замовлення товару; - можливість перегляду списку замовлень; - можливість перегляду товару.	- розбиття товарів на групи та підгрупи; - підтримка смартфонів та планшетів; - простий інтерфейс; - можливість покупки в розстрочку - можливість зарезервувати товар.

Проаналізувавши усі переваги і недоліки вище розглянутих систем, можна виділити основні характеристики програмної системи інтернет-магазину продажу програмного забезпечення:

Функціональні:

- можливість додавання фотографій товару;
- можливість вивантажити товари на сайт магазину;
- можливість скласти заявку на замовлення товару;
- можливість прикріплення необмеженої кількості файлів до опису товару;
- можливість вибору та додавання товару на сайт;
- можливість перегляду списку замовлень;
- можливість перегляду товару. Нефункціональні:
- захищеність;
- адаптивність.
- розбиття товарів на групи та підгрупи;
- підтримка смартфонів та планшетів;
- простий інтерфейс;
- можливість перегляду товару;
- можливість покупки в розстрочку;
- можливість комунікації менеджера з клієнтами;
- можливість паралельної роботи необмеженої кількості користувачів по мережі;
- заборона на перегляд собівартості продавцями.

Рекомендована література

1. Гради Буч Объектно-ориентированный анализ и проектирование с примерами приложений на С++. – СПб.: «Невский диалект», 2001. – 569с.
2. Елізабет Халл, Кен Джексон, Дік Джеремі, “Інженерія вимог” (Requirements Engineering), пер. ДМК Пресс, 2017, 224 с.
3. Люшенко Л.А., Хицко Я.В.2020, Розробка та аналіз вимог до програмного забезпечення [Онлайн ресурс] – Режим доступу: https://ela.kpi.ua/bitstream/123456789/38101/1/Rozrobka_ta_analiz_KP.pdf
4. Соммервілл Іан. Інженерія програмного забезпечення (рос. мовою), 6-е издание: Пер. с англ. – М., 2002.
7. Standard for Software Verification and Validation Plans (ANSI/IEEE standard 1012-1986)
8. Guide to Software Engineering Base of Knowledge (SWEBOK [Онлайн ресурс] – Режим доступу: sorlik.blogspot.com/
9. Співак І.Я., Крепич С.Я., Дарморост І.А. Методичні рекомендації до виконання лабораторних робіт з дисципліни «Аналіз вимог до програмного забезпечення» [Онлайн ресурс] – Режим доступу: http://dspace.wunu.edu.ua/bitstream/316497/36291/1/%D0%9C%D0%92%20%D0%BB%D0%B0%D0%B1%20%D0%BF%D0%BE%20%D0%90%D0%92%D0%9F%D0%97.pdf/
10. ГОСТ 34.201-89 «Виды, комплектность и обозначение документов при создании автоматизированных систем».
11. ГОСТ 19.201-78 (СТ СЭВ 1627-79) Техническое задание. Требования к содержанию и оформлению».
12. ГОСТ 2.102-68 «Виды и комплектность конструкторских документов».
13. ДСТУ 3973-2000 «Система розроблення та поставлення продукції на виробництво».
14. ГОСТ 2.103-2013 «Единая система конструкторской документации. Стадии разработки».
15. ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы».

Лабораторна робота №2

«Дослідження процесів аналізу вимог до програмного забезпечення: концепція створення програмного забезпечення; розробка глосарію; визначення основних акторів; формулювання варіантів використання системи»

Мета роботи: Засвоєння базових знань щодо опису основних позицій концепції створення програмного забезпечення; розробки глосарію; визначення основних акторів; формулювання варіантів використання розроблюваної інформаційної, інформаційно-комунікаційної або програмної системи.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Ознайомитись з діючими стандартами.
4. Дослідити основні принципи опису процесів і визначення вимог до автоматизованих систем.
5. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
6. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

2 Назва та мета роботи.

2. Коротко теоретичні відомості

6. Завдання дослідження лабораторної роботи згідно до обраного варіанта.

7. Результати виконаного дослідження.

8. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Приклад виконання лабораторної роботи з покроковим описом завдань

і. Концепція створення програмного забезпечення

Завдання 1.1. Визначення основних положень концепції

Мета

Зібрання, аналіз і визначення високорівневих потреб і можливостей системи інтернет-магазину продажу програмного забезпечення. Документ акцентує увагу на можливостях, необхідних приватним підприємцям і цільовим користувачам ПК і на тому, чому ці потреби існують.

Контекст

Цей документ розробляється в рамках проекту автоматизації діяльності інтернет-магазину продажу ПЗ.

Визначення і скорочення

Основні визначення приведені в «Глосарій проекту».

Посилання

Бачення базується на документі «Публічна оферта».

Короткий зміст

Цей документ регламентує умови та порядок надання Послуг інтернет-магазином, для здійснення замовлення й придбання Покупцем Товару через мережу Інтернет, які надаються Виконавцем, а також права і обов'язки, що виникають у зв'язку з цим у Покупця, Виконавця й Одержувача.

Завдання 1.2. Позиціонування продукту на ринку

Ділові переваги

У даний час продаж програмного забезпечення в інтернет–магазинах здійснюється на основі використання електронних таблиць. Порівняно з тим, що є, нове рішення забезпечить зручніший режим доступу зацікавлених осіб до

інформації, підвищить швидкодію, забезпечить надійне зберігання даних і повніший набір функцій, що підлягають автоматизації.

Визначення проблеми

Визначення проблем приведене у таблиці 1.1

Таблиця 1.1. Визначення проблем

Проблема	**Недостатня швидкість відповіді сервера на запити**
Стосується	Адміністратора
Її наслідком є	Довго завантажується сайт
Успішне рішення	Стиснути або видалити зображення з високою роздільною здатністю
Проблема	*Биті посилання*
Стосується	Адміністратора
Її наслідком є	Користувач натискає на посилання та переходить на сторінку помилки 404, 303, 302
Успішне рішення	Забезпечити хороший UX, оновити посилання, видалити з індексації пошукової системи ті, які ведуть на сторінку 404, 302, 303
Проблема	*Застарілий дизайн сайту*
Стосується	Адміністратора
Її наслідком є	Великий відсоток відмов
Успішне рішення	Змінити застарілий дизайн сучасними стандартами веб-дизайну
Проблема	*Недостатньо оперативний обмін інформацієюміж адміністратором та менеджером з продаж*
Стосується	Адміністратора, менеджера з продаж
ЇЇ наслідком є	Затримка у роботі інтернет-магазину
Успішне рішення	Оптимальна організація роботи сайту, економія часу всіх учасників процесу

Визначення позиції продукту

Визначення позиції продукту представлене у таблиці 1.2.

Таблиця 1.2. Визначення позиції продукту

Для	Магазин продажу ПЗ
Якому	Потрібно автоматизувати процес продажу програмного забезпечення
(Назва продукту)	ПС «**SOFTWARE**»
Який	Заснований на промисловій СУБД і високонадійний
На відміну від	Існуючого механізму на основі електронних таблиць

Наш продукт	Виключає помилки продажу програмного забезпечення

Завдання 1.3. Опис користувачів програмного продукту

Відомості про користувачів

У системі існують два основі користувачі: адміністратор (власник), менеджер. Менеджер – вводить дані про замовлення товару для інтернет-

магазину, контролює їх виконання, приймає замовлення у покупців та веде статистику продажу. Адміністратор – контролює усі дії менеджера, виправляє помилки системи та оптимізує роботу сайту.

Профілі користувачів

У таблиці 1.3 представлено профілі користувачів.

Таблиця 1.3. Профілі користувачів

Типовий представник	**Адміністратор**
Опис	Користувач системи, наділений правами адміністратора, у нього є доступ до усіх функцій
Тип	Адміністратор
Відповідальності	Надсилати певні вказівки іншим користувачам системи, виправляти помилки системи, оптимізувати роботу системи.
Критерій успіху	Можливість прогнозувати прибуток і популярність інтернет-магазину, відповідно статистики продаж;
Типовий представник	**Менеджер продаж**
Опис	Користувач системи, наділений правами на зміну планової інформації у системі
Тип	Користувач
Відповідальності	Ведення статистики стосовно проданого товару; підтримка бази даних постійних клієнтів; контроль товарообігу;
Критерій успіху	Наявність в БД інформації про постійних клієнтів; статистика товарообігу доступна для власника;

Завдання 1.4. Опис можливостей програмного продукту

Можливість опису замовлення через впорядковану в часу сукупність робіт, а також параметрів.

Можливість отримати інформацію про те, через скільки часу буде доставлене замовлення.

Можливість ведення статистики і формування звітності продаж.

Можливість обмінюватись повідомленнями в програмному середовищі між користувачами, що наразі знаходяться в системі.

Можливість редагування бази даних постійних клієнтів, видача різного роду бонусів в залежності від уже куплених товарів.

Можливість редагувати будь-яку інформацію, наприклад: є в наявності, немає; зміна ціни; зміна опису та ін.

Завдання 1.5. Визначення обмежень застосування родукту провадження системи не повинне займати більше 3 місяців. У ядрі системи повинна бути представлена промислова СУБД.

Завдання 1.6. Визначення показників якості програмного продукту (застосовуваність, надійність)

Застосовуваність

- Час, необхідний для навчання звичайних користувачів, - 16 годин, для навчання досвідчених користувачів – 4 години.
- Час відгуку для типових завдань – не більше 5 секунд, для складних завдань – не більше 20 секунд.

Надійність

- Доступність – час, що витрачається на обслуговування системи, не повинен перевищувати 2% від загального часу роботи.
- Середній час безвідмовної роботи – 15 робочих днів.
- Максимальна норма помилок або дефектів – 1 помилка на двадцять тисяч рядків коду.

Завдання 1.7. Визначення інших вимог до продукту (вживані стандарти, системні вимоги та експлуатаційні вимоги)

Вживані стандарти

Система повинна відповідати всім стандартам інтерфейсу користувача Microsoft® Windows®.

Системні вимоги

Мінімальні системні вимоги:

- 128 Mb пам’яті;

- 200 Mb вільного дискового простору;
- Процесор з тактовою частотою 900 MHz;
- Операційна система Windows XP/7/8/8.1

Експлуатаційні вимоги

Система повинна бути здатна підтримувати мінімум 15 одночасно запитів користувачів, пов'язаних із загальною базою даних і мати можливість їх подальшого збільшення.

Завдання 1.8. Загальні вимоги до документації

Керівництво користувача

У системі повинно бути представлене Керівництво користувача (по типам користувачів). Воно повинно містити розшифровку всіх використовуваних термінів, описи основних варіантів використання, включаючи альтернативні сценарії, а також докладний огляд інтерфейсу програми

Інтерактивна довідка

Інтерактивна довідка необхідна для вирішення питань, які виникли під час роботи. У довідці повинна бути реалізована можливість пошуку інформації за ключовими словами, а також варіант представлення інформації по окремих позиціях меню програми. Довідка повинна містити максимально повну і докладну інформацію по роботі системи.

Керівництво по установці і конфігурації

Система повинна мати керівництво по установці у файлі ReadMe.txt, який повинен додаватися до системи. Файл ReadMe.txt повинен містити докладну інструкцію по установці даної системи, щоб у разі потреби користувач зміг провести установку самостійно без допомоги адміністратора.

2. Приклад розробки глосарію для програмної системи «Інтернет- магазин продажу програмного забезпечення»

Інтернет-магазин – вітрина, розташована в Інтернет мережі для розміщення, рекламування і продажу товарів, запропонованих виробниками.

Лендінг – сайт (сторінка) в Інтернет мережі за адресою (посиланням), яка використовується власником для надання послуг виробникам товарів з розміщення, рекламування та їх продажу в інтернет-магазині.

Товар – пристрій, виріб, засіб, аксесуар, а також інша продукція побутового вжитку, представлена виробником (постачальником) цього

товару для продажу, який рекламується та реалізовується через вітрину інтернет- магазину за допомогою послуг системи.

Система – програмний комплекс (автоматизована інформаційна система) представлена інтернет-магазином покупцю, для дистанційного, спрощеного перегляду товарів за допомогою мережі Інтернет, де покупцю надається можливість ознайомитись з технічними характеристиками, інструкцією і властивостями зацікавленого товару, а також процедурою його замовлення, придбання та отримання.

Замовлення – заявлена замовником необхідність у виготовленні або придбанні продукції.

Робота – одиниця попереднього планування. Являє собою роботу на конкретному обладнанні (*ресурсі*) над однією одиницею продукції (замовленням).

Ресурс – одиниця обладнання або виконавець.

3. Приклад виконання лабораторної роботи з покроковим описом завдань

Завдання 3.1. Виявлення акторів.

Необхідно виявити максимально можливу кількість акторів (це можуть бути потенційні користувачі системи, а також зовнішні системи). Здійснити класифікацію акторів, розбити їх на групи; сформулювати кінцевий реєстр акторів. Кожному актору дати короткий (в один абзац) опис.

На рисунку 3.1 представлені основні кандидати в актори системи.

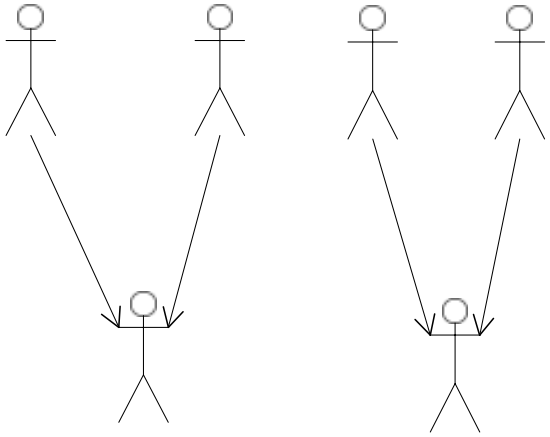
Інтерв'ю, яке було проведено з вказаними вище кандидатами показало, що менеджер по продажу та старший менеджер припускають використовувати програмну систему, що розробляється однотипно. Це дозволило узагальнити ці дві ролі в одну. Аналогічна ситуація з продавцями, оскільки вони всі виконують однакові операції. Короткий опис акторів представлено в таблиці 3.1.

Таблиця 3.1. Короткий опис акторів

Актор	**Короткий опис**
Адміністратор	Слідкує за усіма процесами, надсилає вказівки іншим користувачам системи, прогнозує прибутки та витрати магазину
Менеджер	Слідкує за процесом обліку товарів, проводить ревізію, аналізує роботу продавця, замовляє нові товари

Продавець	Проводить облік товарів, веде статистику продажу, реєструє будь-які зміни в кількості, чи інших показниках товару

Адміністратор Менеджер по продажу Старший менеджер Продавець №1 Продавець №2



Менеджер Продавець

Рис. 3.1. Аналіз акторів системи

Завдання 3.2. Виявити варіанти використання

Для кожного з акторів необхідно виявити максимально можливу кількість варіантів використання. Кожному з них дати коротке (в одно речення) формулювання. Виявлені варіанти використання зведені в таблицю 3.2.

Таблиця 3.2. Виявлення варіантів використання

Основний актор	**Найменування**	**Формулювання**
Продавець	Реєстрація товару	Цей варіант дозволяє продавцю додати у систему новий товар
Продавець	Вилучення товару	Цей варіант дозволяє продавцю вилучити певну кількість товару з системи

Продавець	Формування звіту	Цей варіант дозволяє продавцю створити вихідний документ у системі, який показує яку кількість товару він продав за цей день
Менеджер	Перегляд звітності	Цей варіант дозволяє менеджеру переглянути звіти, які створили продавці
Менеджер	Реєстрація товару	Цей варіант дозволяє менеджеру додати у систему новий товар
Менеджер	Замовлення товару	Цей варіант дозволяє менеджеру замовити новий товар, та уточнити деталі про доставку
Адміністратор	Інформаційне повідомлення	Цей варіант дозволяє адміністратору надіслати будь-яке повідомлення будь-якому користувачу у будь-який момент часу
Адміністратор	Перегляд звітності	Цей варіант дозволяє адміністратору переглядати звіти як продавців так і менеджерів магазину
Адміністратор	Запит про статистику	Цей варіант дозволяє переглянути статистику того, що було зроблено за будь-який день, кожним користувачем системи

Завдання 3.3. Скласти діаграми варіантів використання

Складається одна загальна або декілька часткових діаграм варіантів використання. Загальні вимоги: кожний варіант використання і кожний актор повинні бути відображені хоча б на одній діаграмі. Всі варіанти використання показані на рис. 3.2.

Завдання – атомарна одиниця планування диспетчером. В процесі попереднього планування кожній роботі відповідає своє завдання. Завдання асоціюється з однією одиницею обладнання, працівником або групою.

Послуга – комплекс дій, які вчиняються виконавцем відповідно до оформленого покупцем за допомогою системи інтернет-магазину замовлення, для подальшої взаємодії та реалізації.

Виконавець(менеджер) – фізична особа (фізична особа-підприємець) або юридична особа, яка діє в інтересах інтернет-магазину на підставі укладених договорів, угод, доручень, виданих довіреностей тощо, для здійснення таких операцій як: обробка замовлень, розрахунків, прийому-передачі чи відправки придбаних (замовлених) Товарів.

Покупець – фізична або юридична особа, яка має намір придбати товар через мережу Інтернет використовуючи систему та послуги представлені інтернет-магазином на умовах дотримання цієї оферти.

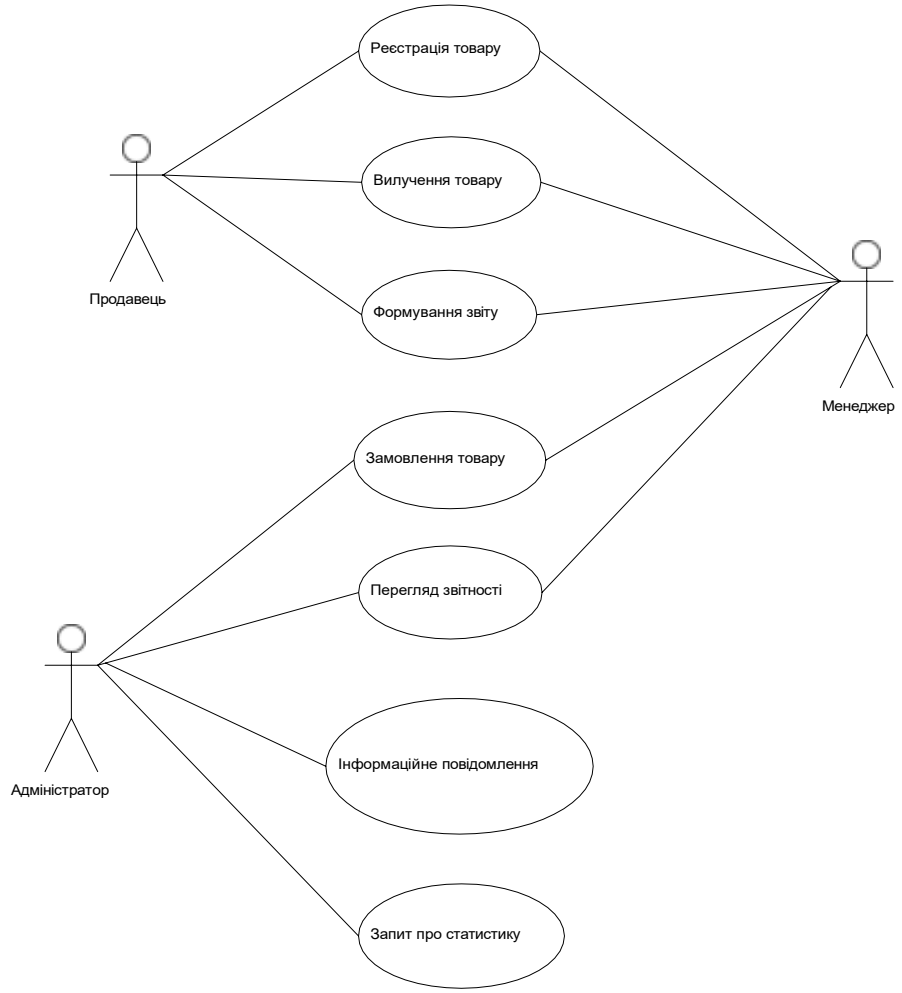


Рис. 3.2. Діаграма прецедентів системи

Оферта – договір, який регулює відносини продажу товарів побутового вжитку, що замовляються та придбаються через мережу Інтернет за допомогою вітрини та комплексу послуг, які надаються інтернет-магазином.

Виробник – суб'єкт господарювання, який: виробляє товар або заявляє про себе як про виробника товару чи про виготовлення такого товару на замовлення, розміщуючи на товарі та/або на упаковці чи супровідних документах, що разом з товаром передаються споживачеві своє найменування, торговельну марку або інший елемент, який ідентифікує такого суб'єкта господарювання; або імпортує товар і несе відповідальність

за виробництво, поставку та забезпечення відповідності кожного виготовленого нею товару.

Постачальник – фізична або юридична особа яка на підставі укладених цивільно-правових угод, договорів, сертифікатів офіційного представника (дилера) здійснює доставку товарів до пункту призначення.

Одержувач – покупець, або уповноважена ним фізична чи юридична особа, яка на підставі відповідних документів (довіреність, договір) має право отримати придбаний покупцем товар в точці продажу, а також володіє правом

провести його обмін чи повернути його з підстав передбачених цією офертою та діючим законодавством України.

Менеджер з продаж – атомарний людський ресурс при плануванні.

Статус роботи – стан роботи з точки зору продаж. В першому наближенні розрізняють наступні статуси: «робота замовлення, прийнятого до обробки», «робота частково виконана», «робота виконана».

Допустимий інтервал – терміни, в які може бути виконана (запланована) робота.

Критичний термін виконання робіт – термін початку роботи в плані, перенесення роботи пізніше якого призводить до зриву строків відправлення замовлення.

Колізія – суперечлива інформація в плані, яка приводить або до неможливості виконання плану, або до порушення обов'язків перед замовником. Колізії можливі у випадках:

1) Була вказана застаріла ціна;

2) Неправильно вказані дати виконання замовлення.

Лабораторна робота №3

«Дослідження процесів розробки діаграм станів та активності розроблюваниз програмних систем»

Мета роботи: Засвоєння базових знань щодо опису розробки діаграм станів, діаграми активності розроблюваної системи.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Дослідити основні принципи опису процесів і визначення вимог до автоматизованих систем.
4. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
5. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

3 Назва та мета роботи.

2. Коротко теоретичні відомості

9. Завдання дослідження лабораторної роботи згідно до обраного варіанта.

10. Результати виконаного дослідження.

11. Висновки по роботі.

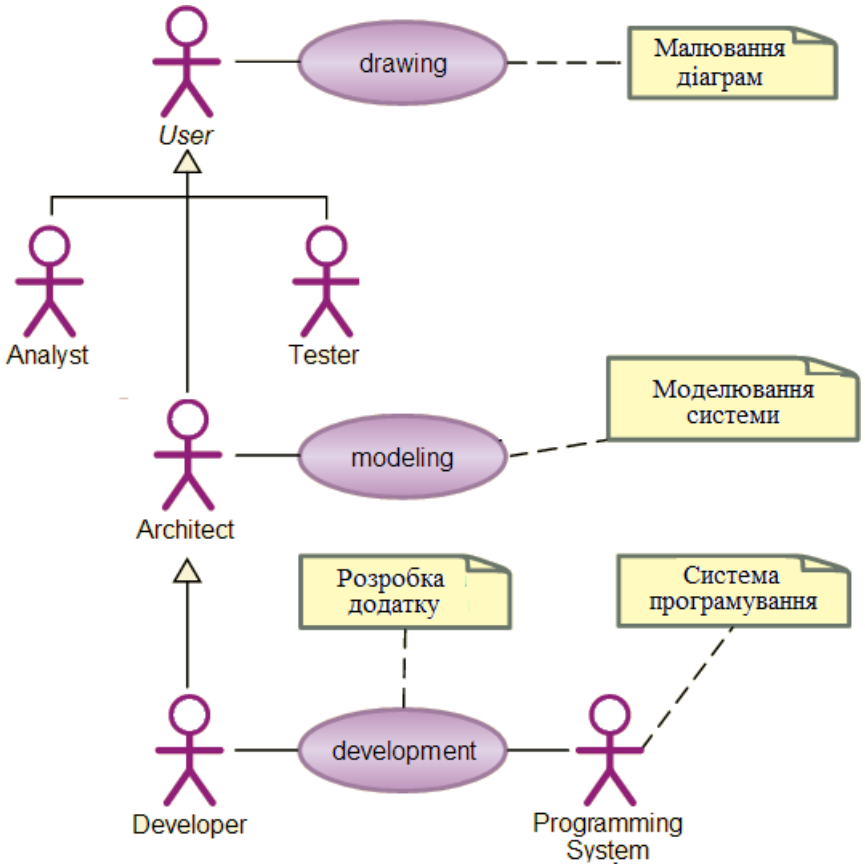
Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Розробка UML-діаграм розроблюваної системи: діаграми станів та діаграми активності

Можна виділити три основні варіанти використання UML



Варіант використання **drawing** («Малювання діаграм») має на увазі *зображення діаграм UML* з метою обдумування, обміну ідеями між людьми, документування і тому подібного. Значимим для користувача **User** результатом в цьому випадку є саме зображення діаграм. Іноді малювання діаграм від руки фломастером з подальшим фотографуванням цифровим апаратом може виявитися практичним.

Варіант використання **modeling** («Моделювання систем») має на увазі *створення і зміну моделі* системи в термінах тих елементів моделювання, які передбачаються метамоделлю UML. Значимим результатом в цьому випадку є **машинно-читаний артефакт** з описом моделі. Скорочено називатимемо такий артефакт просто ***моделлю***, діяльність зі складання моделі називатимемо моделюванням, а суб'єкта моделювання називатимемо архітектором Architect.

Варіант використання **development** («Розробка додатків») має на увазі детальне моделювання, реалізацію і тестування додатка в термінах UML.

Значимим для користувача **Developer** результатом в цьому випадку є працюючий додаток, який може бути скомпільований в мову, підтримувану конкретною системою програмування або відразу інтерпретований середовищем виконання інструменту. Цей варіант використання найскладніший в реалізації.

Сучасні інструменти підтримують вказані варіанти використання далеко не в рівній мірі. Усі інструменти уміють (погано або добре) візуалізувати усі типи діаграм UML, деякі інструменти дозволяють побудувати модель, що допускає якесь подальше використання, але тільки небагато інструментів можуть генерувати виконуваний код і то, ні в якому разі, не для усіх діаграм.

В діаграмах краще використати кольори (для наочності).

Наприклад, тут будемо додержуватися певної палітри кольорів, що відображає вибране представлення моделі:

 – моделювання використання (бузковий);

 – моделювання структури (жовтий);

 – моделювання поведінки (блакитний);

 – елементи, використовувані і для моделювання структури, і для моделювання поведінки (зелений);

 – стереотипи, обмеження і інші значимі елементи (сірий).

UML для бізнес-моделювання

Unified Modeling Language (UML) – уніфікована мова моделювання. Розшифруємо: modeling має на увазі створення моделі, що описує об'єкт. Unified (універсальний, єдиний) - підходить для широкого класу проектованих програмних систем, різних галузей додатків, типів організацій, рівнів компетентності, розмірів проектів. UML описує об'єкт в єдиному заданому синтаксисі, тому де б ви не намалювали діаграму, її правила будуть зрозумілі для всіх, хто знайомий із цією графічною мовою — навіть в іншій країні.

Одне із завдань UML — служити засобом комунікації всередині команди та спілкування із замовником. Розгляньмо можливі варіанти використання діаграм.

- Проектування. UML-діаграми допоможуть при моделюванні архітектури великих проектів, в якій можна зібрати як великі, так і дрібніші деталі і намалювати каркас (схему) програми. По ньому згодом будуватиметься код.
- Реверс-інжиніринг — створення UML-моделі з існуючого програмного коду, зворотна побудова. Може застосовуватися, наприклад, на проектах підтримки, де є написаний код, аледокументаціянеповна чи відсутня.
- З моделей можна витягувати текстову інформацію і генерувати відносно тексти, що легко читаються, — документувати. Текст та графіка доповнюватимуть один одного.

Як і будь-яка інша мова, UML має власні правила оформлення моделей та синтаксис. За допомогою графічної нотації UML можна візуалізувати систему, поєднати всі компоненти в єдину структуру, уточнювати та покращувати модель у процесі роботи. На загальному рівні графічна нотація UML містить 4 основні типи елементів:

- фігури;
- лінії;
- значки;
- написи.

UML-нотація є де-факто галузевим стандартом у галузі розробки програмного забезпечення, IT-інфраструктури та бізнес-систем.

У мові UML є 12 типів діаграм:

- 4 типи діаграм представляють статичну структуру програми;
- 5 типів становлять поведінкові аспекти системи;
- 3 становлять фізичні аспекти функціонування системи (діаграми реалізації).

Деякі види діаграм специфічні для певної системи та програми. Найдоступнішими з них є:

- діаграма прецедентів (Use-case diagram);
- діаграма класів (Class diagram);
- діаграма активностей (Activity diagram);
- Діаграма послідовності (Sequence diagram);
- Діаграма розгортання (Deployment diagram);
- діаграма співпраці (Collaboration diagram);
- діаграма об'єктів (Object diagram);

▪ Діаграма станів (Statechart diagram).

Діаграма прецедентів - Use-case diagram

Діаграма прецедентів використовує 2 основних елементи:

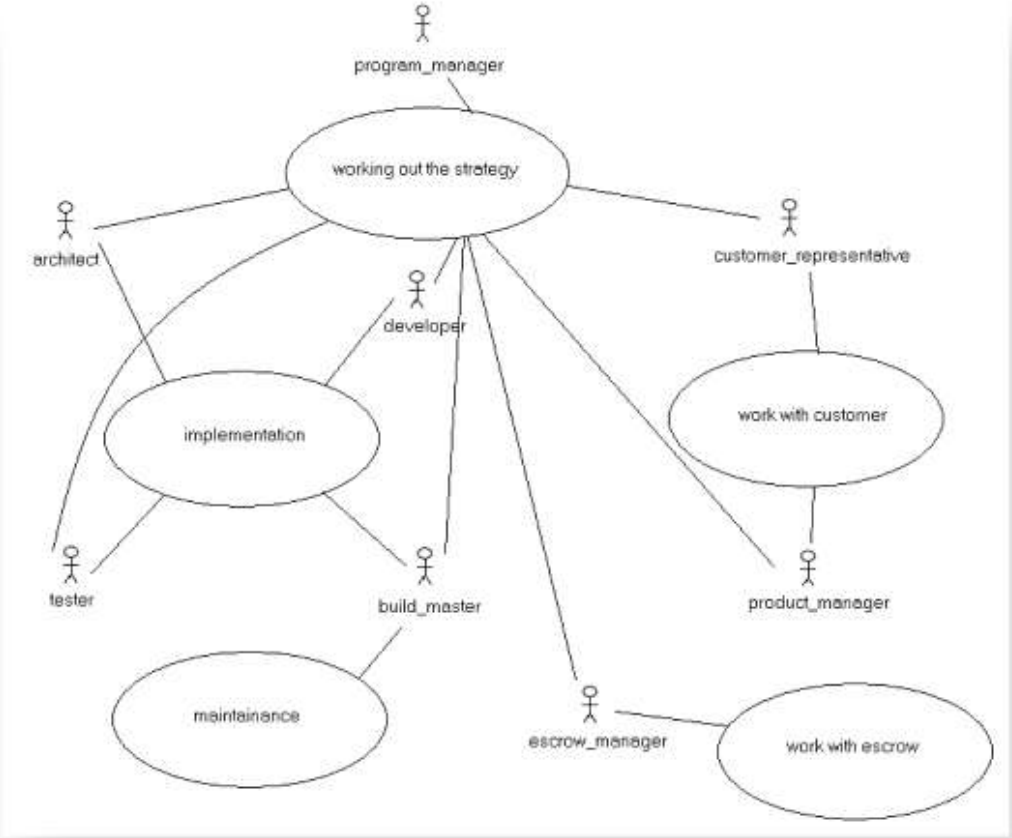
1) Actor (учасник) – безліч логічно пов'язаних ролей, що виконуються при взаємодії з прецедентами чи сутностями (система, підсистема чи клас). Учасником може бути людина, роль людини в системі або інша система, підсистема чи клас, які є чимось поза сутністю.

2) Use case (прецедент) - опис окремого аспекту поведінки системи з погляду користувача. Прецедент не показує, як досягається певний результат, а тільки що саме виконується.

Розглянемо класичний студентський приклад, у якому є 2 учасники: студент та бібліотекар. Прецеденти для студента: шукає у каталозі, замовляє, працює у читальному залі. Роль бібліотекаря: видача замовлення, консультації (рекомендації книг на тему, навчання використанню пошукової системи та заповнення бланків замовлення).

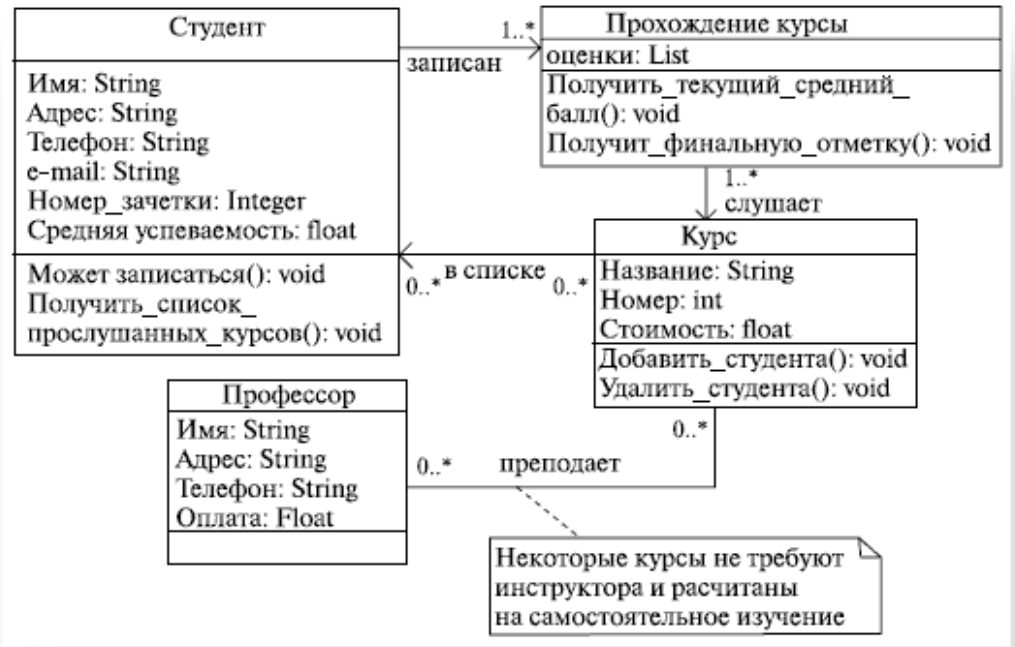
Другий приклад трохи складніший. Бачимо, що одна й та сама особа може виступати в декількох ролях. Наприклад, product manager у нас працює над стратегією та більше нічим не займається, архітектор працює над стратегією та займається впровадженням, build master займається трьома

речами одночасно, і так далі. За такою схемою ми можемо простежити, яка ролей пов'язана з якими прецедентами.



Діаграма класів - Class diagram

Клас (class) – категорія речей, які мають спільні атрибути та операції. Сама діаграма класів є набір статичних, декларативних елементів моделі. Вона дає нам найповніше і розгорнуте уявлення про зв'язки в програмному коді, функціональність та інформацію про окремі класи. Програми генеруються найчастіше саме з діаграми класів. Розглянемо на простому прикладі нижче:



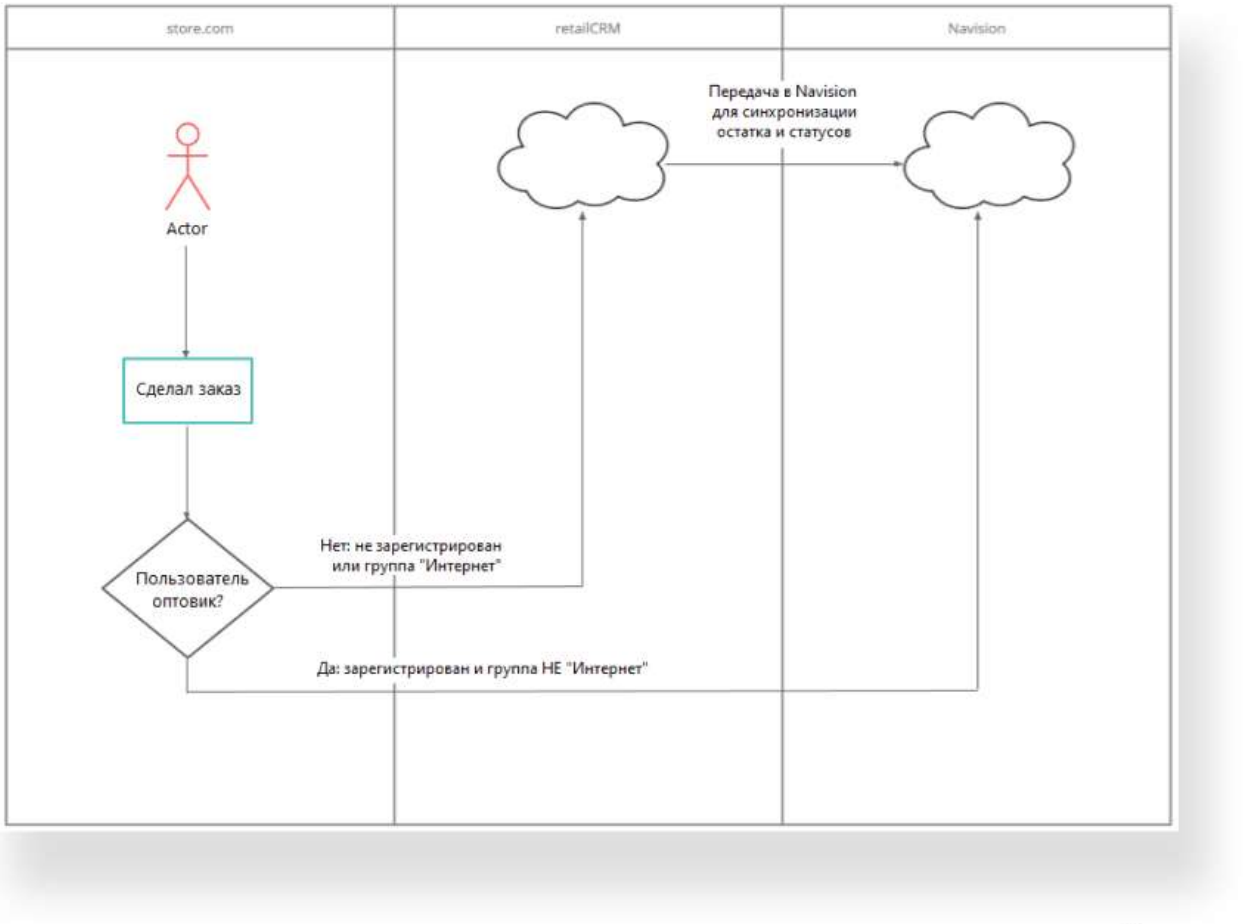
Для класу "студент" є таблиця, що містить атрибути: ім'я, адресу, телефон, e-mail, номер заліковки, середня успішність. І також показані зв'язки цієї сутності з іншими: проходженням курсу, який слухає, хто професор. У цьому прикладі додаються функції, які можуть бути застосовані до сутності "студент".

Діаграма активностей - Activity diagram

Теж крута штука, яка часто використовується на практиці. Діаграма активностей визначає динамічні аспекти поведінки системи у вигляді блок-схеми, що відображає бізнес-процеси, логіку процедур та потоки робіт – переходи від однієї діяльності до іншої. Власне, ми малюємо алгоритм дій (логіку поведінки) системи чи взаємодії кількох систем. Нижче приклад подібної діаграми для інтернет-магазину.

Діаграма активностей для сайту магазину максимально доступно пояснює які є інтеграції в системі. Актор (у нашому випадку - покупець), який зайшов на сайт, робить замовлення. Далі у нас відбувається розгалуження: перевіряємо, чи користувач оптовиком (Так/Ні). Якщо він не зареєстрований у системі і не оптовик, замовлення надсилається до retailCRM. Якщо користувач зареєстрований, його замовлення потрапляє до

Navision. При цьому між retailCRM та Navision відбувається синхронізація залишку та статусів.



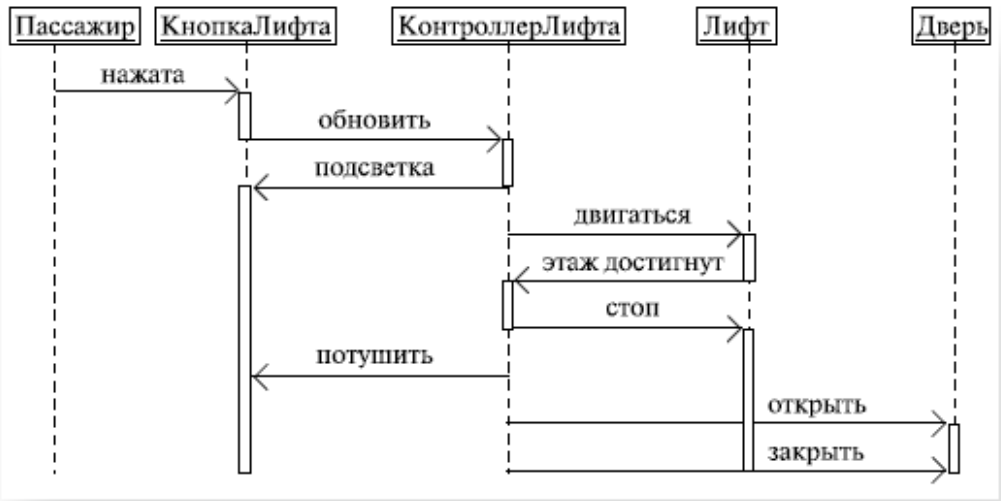
Evergreen

Цю базову діаграму ми можемо доповнити, розширити, вона може бути частиною документації і дає загальне уявлення про роботу системи.

Діаграма послідовності - Sequence Diagram

Використовується для уточнення діаграм прецедентів – визначає поведінкові аспекти системи. Діаграма послідовності відображає взаємодію об'єктів у динаміці, у часі. При цьому інформація набуває вигляду повідомлень, а взаємодія об'єктів передбачає обмін цими повідомленнями в рамках сценарію.

:



Діаграма розгортання - Deployment Diagram

Діаграма розгортання відображає графічне представлення інфраструктури, на яку буде розгорнуто програму: топологію системи та розподіл компонентів її вузлами, а також з'єднання — маршрути передачі даних між вузлами. Діаграма допомагає раціональніше організувати компоненти, від чого залежить серед іншого і продуктивність системи, а також вирішити допоміжні завдання, наприклад, пов'язані з безпекою.

Набір фігур та стрілок може значно спростити вирішення складних завдань у програмуванні, допомогти при виборі оптимального рішення та розробці технічної документації. Які ще висновки можемо зробити:

- будувати діаграми нескладно;
- діаграми дуже легко читаються та прості для розуміння;
- вони - відмінний інструмент для проектування архітектури та поведінки;
- необхідні документування будь-якої нетривіальної системи. Дозволяють легко зрозуміти зв'язки між модулями та інтеграціями у системі.

Приклад побудови діаграм наведено тільки для декількох функцій

На рис. 1 зображена діаграма станів для функції «Пошук програмного забезпечення».

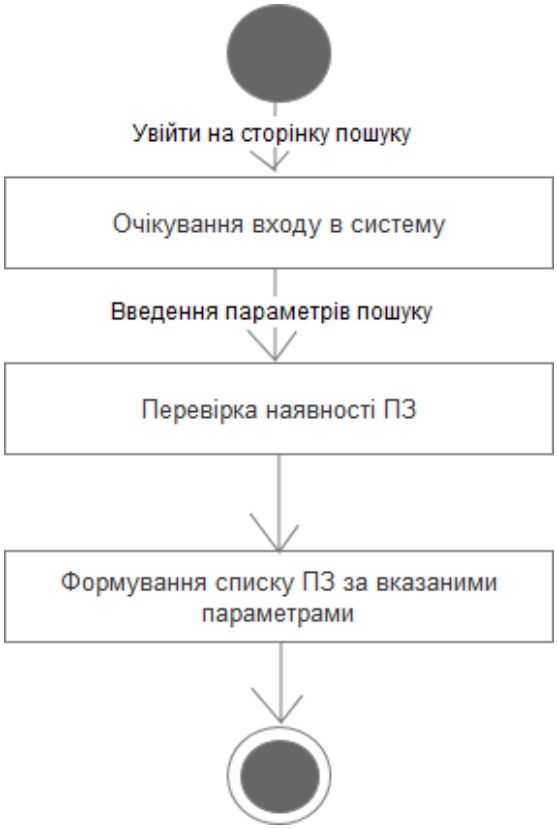


Рис 1. Діаграма станів (пошук ПЗ)

На рис. 2 зображена діаграма станів для функції «Додавання програмного забезпечення».

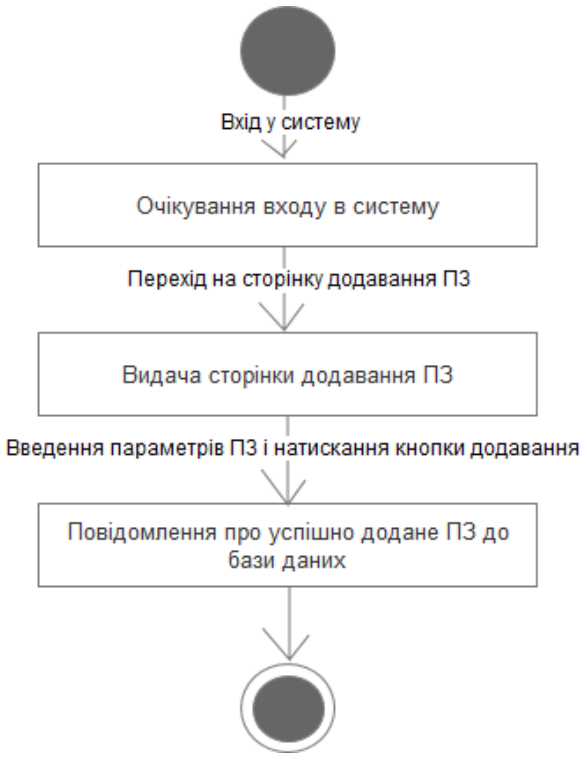


Рис 2. Діаграма станів (додавання ПЗ)

На рис. 3 зображена діаграма станів для функції «Відображення статусу».

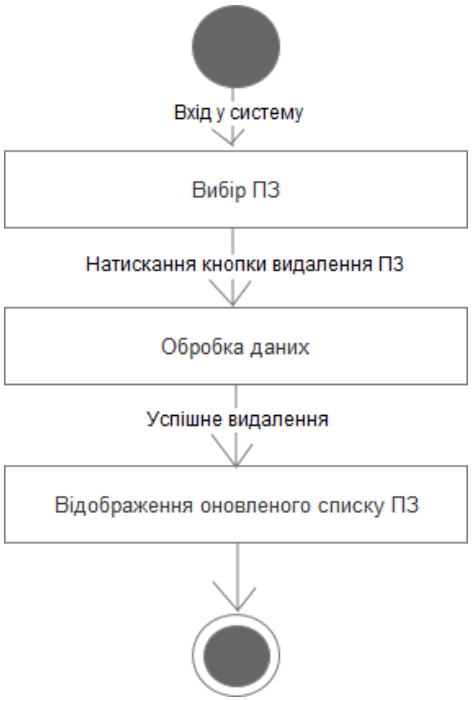


Рис 3. Діаграма станів (Видалення ПЗ)

Діаграма активності для функції «Купівля програмного забезпечення» зображена на рис. 4

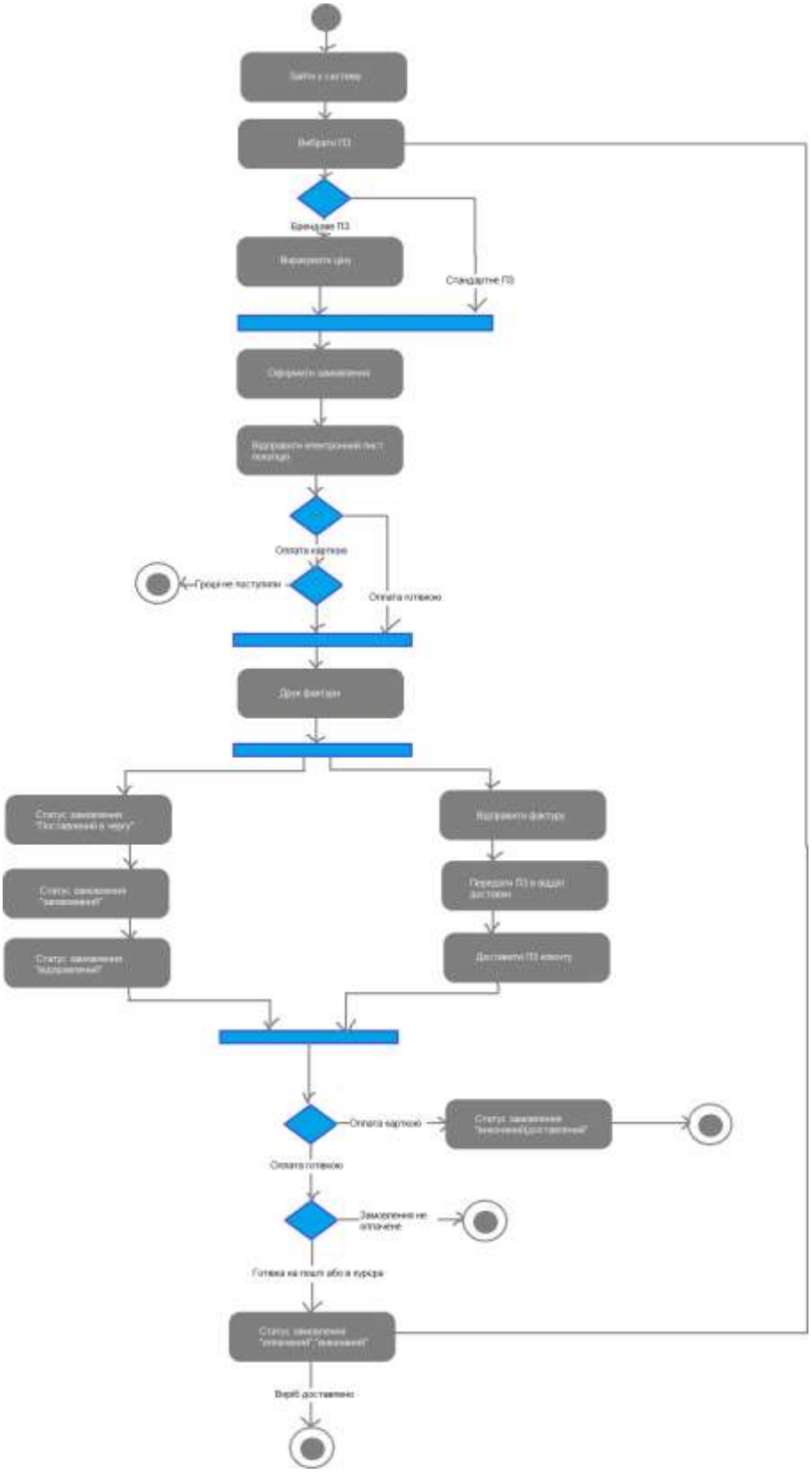


Рис.4. Діаграма активності (Купівля ПЗ)

Приклад розробки прототипу розроблюваної системи

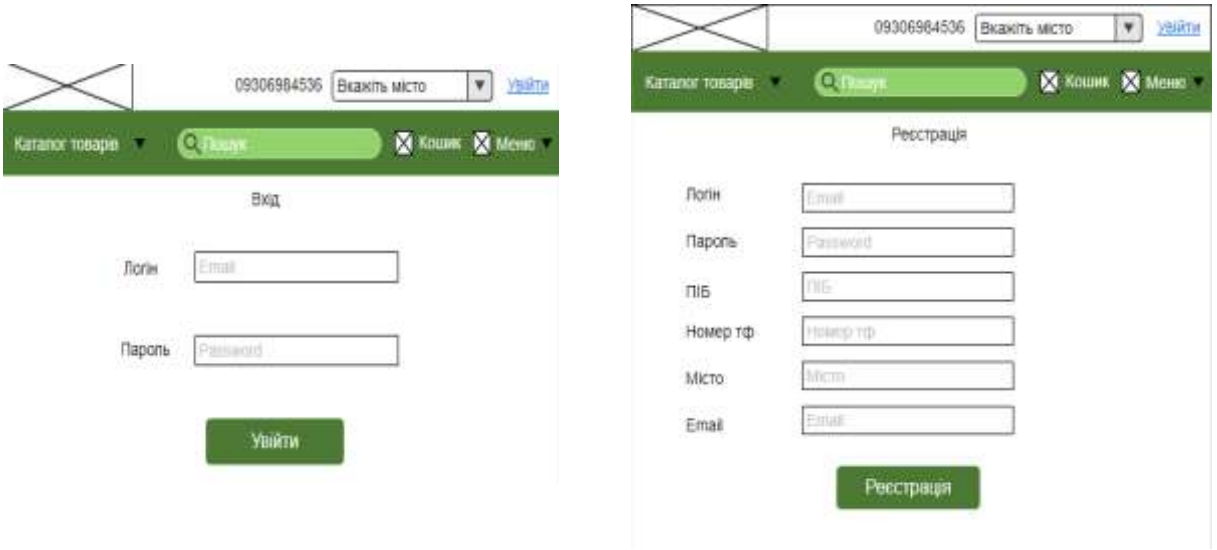


Рис. 5. Прототип головних екранних форм

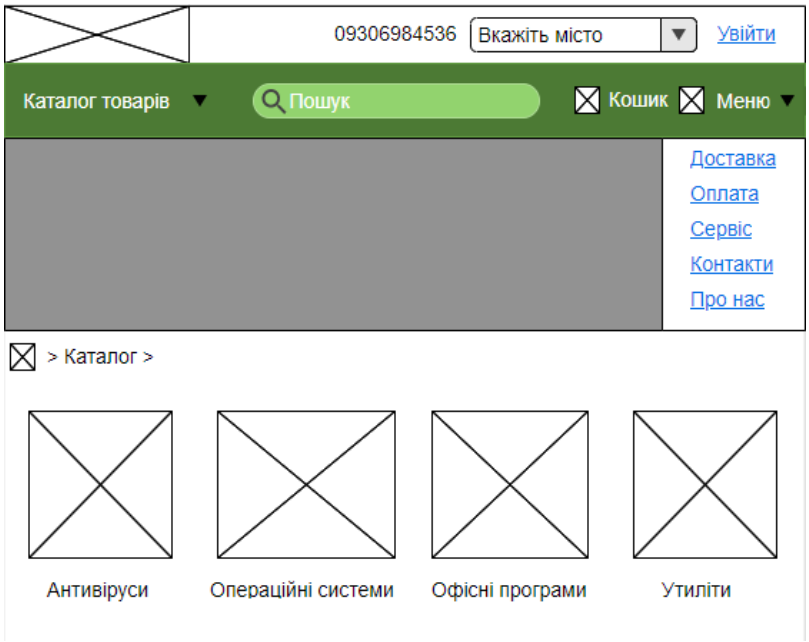


Рис. 6. Прототип екранної форми «Каталог»

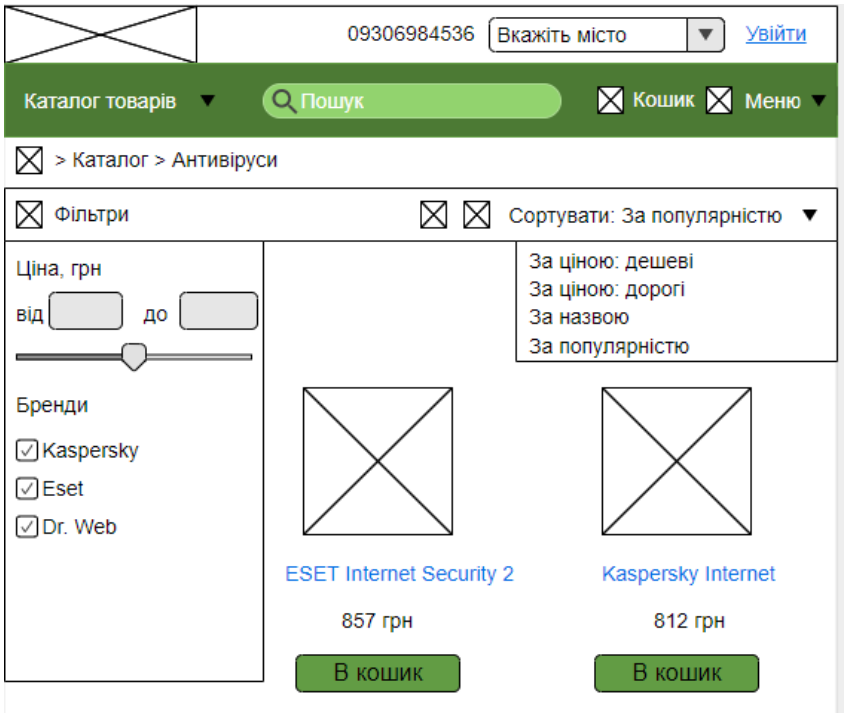


Рис. 7. Прототип екранної форми «Каталог/Антивіруси»

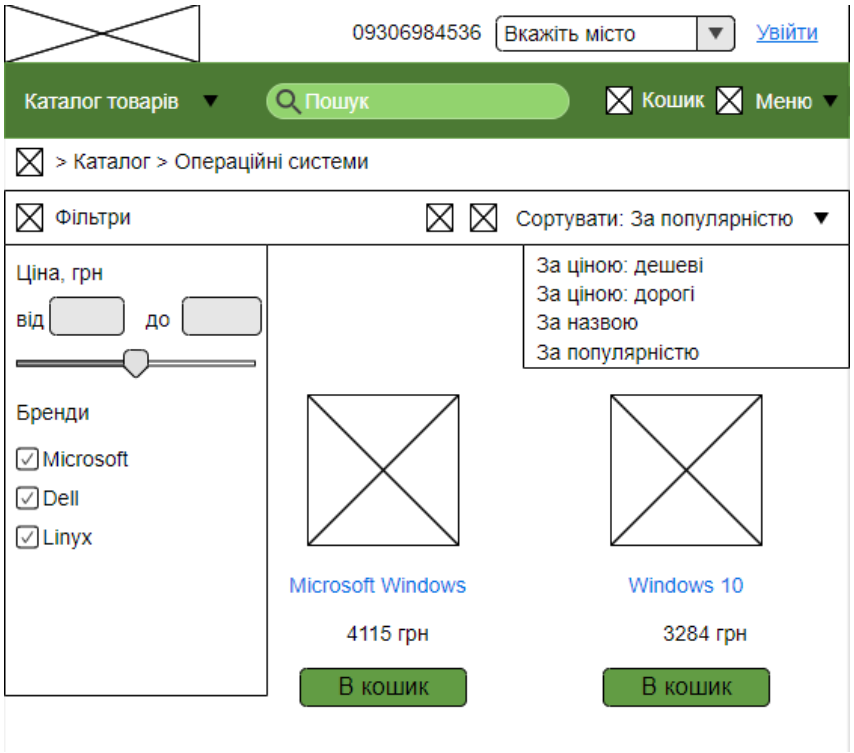


Рис. 8. Прототип екранної форми «Каталог/Операційні системи»

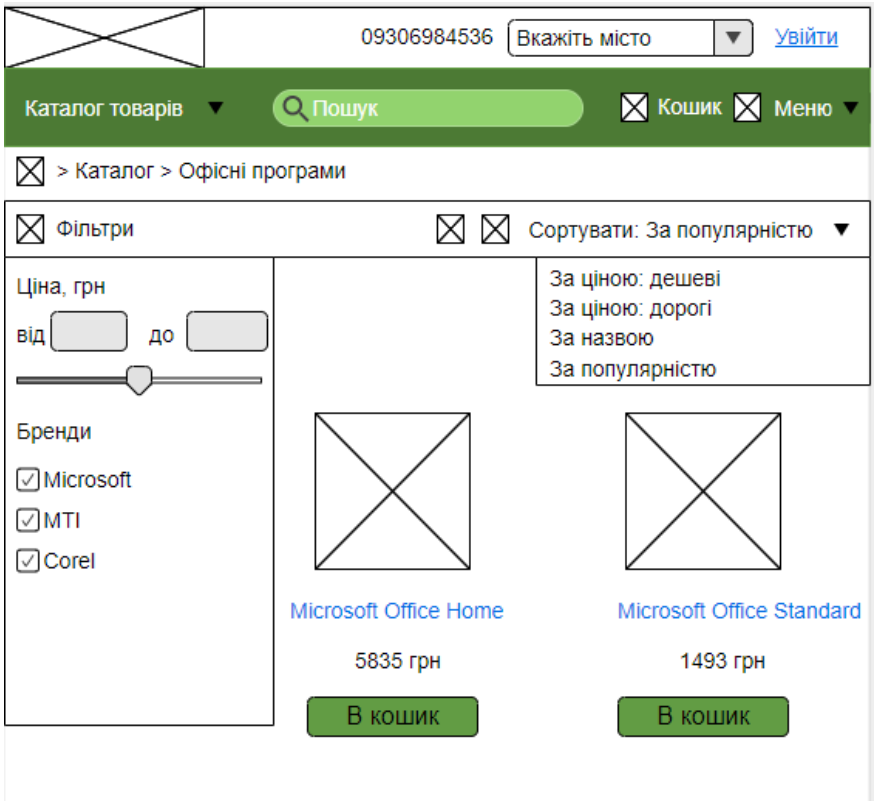


Рис. 9. Прототип екранної форми «Каталог/Офісні програми»

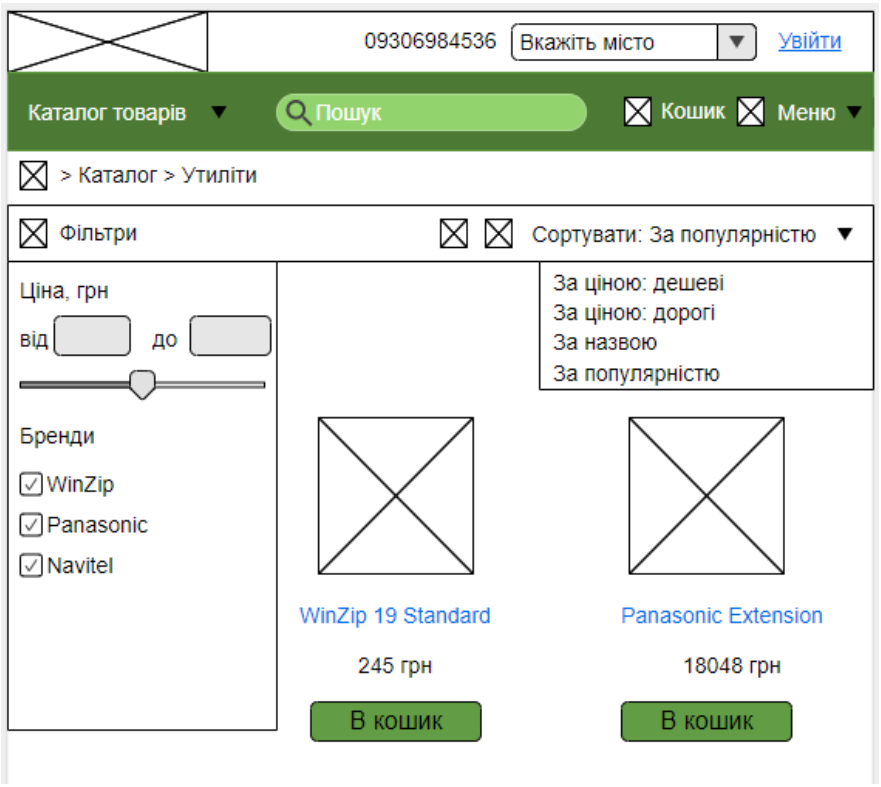


Рис. 10. Прототип екранної форми «Каталог/Утиліти»

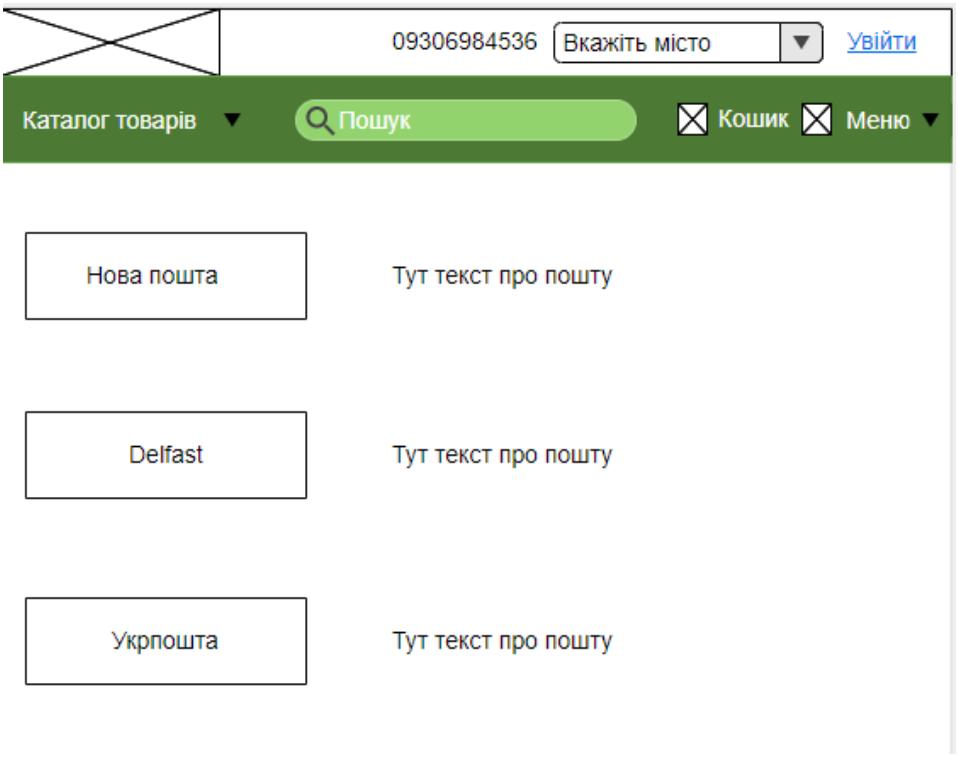


Рис. 11. Прототип екранної форми «Спосіб доставки»

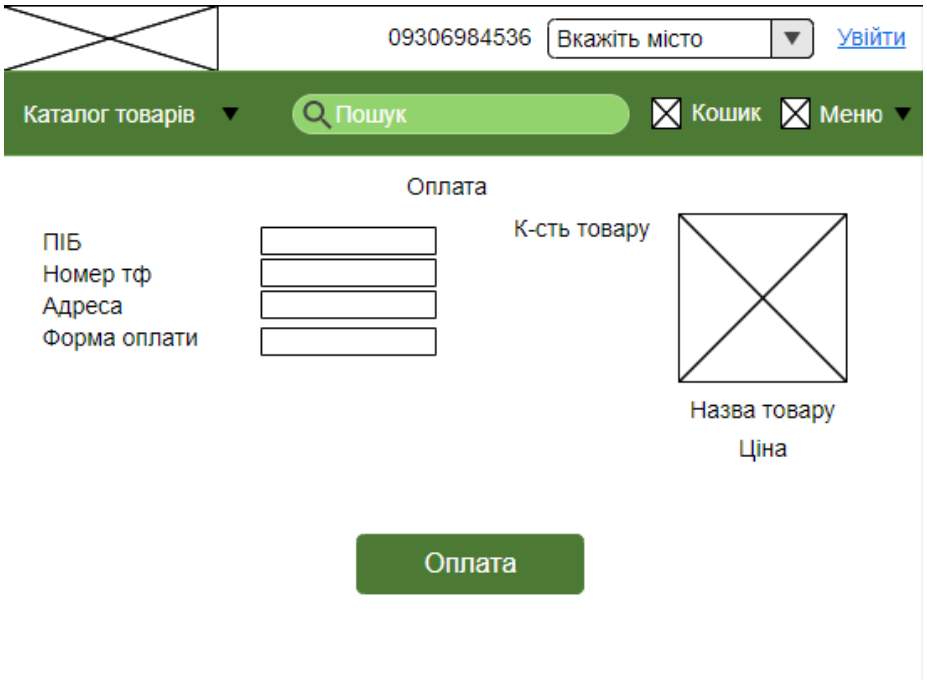


Рис. 12. Прототип екранної форми «Спосіб оплати»

Рекомендована література

1. Гради Буч Объектно-ориентированный анализ и проектирование с примерами приложений на С++. – СПб.: «Невский диалект», 2001. – 569с.

2. Елізабет Халл, Кен Джексон, Дік Джеремі, “Інженерія вимог”

(Requirements Engineering), пер. ДМК Пресс, 2017, 224 с.

3. Соммервілл Іан. Інженерія програмного забезпечення (рос. мовою), 6-е издание: Пер. с англ. – М., 2002.

4. Standard for Software Verification and Validation Plans (ANSI/IEEE standard 1012-1986)

5. Guide to Software Engineering Base of Knowledge (SWEBOK [Онлайн ресурс] – Режим доступу: sorlik.blogspot.com/

6. Співак І.Я., Крепич С.Я., Дарморост І.А. Методичні рекомендації до виконання лабораторних робіт з дисципліни «Аналіз вимог до програмного забезпечення» [Онлайн ресурс] – Режим доступу: http://dspace.wunu.edu.ua/bitstream/316497/36291/1/%D0%9C%D0%92%20%D0%BB%D0%B0%D0%B1%20%D0%BF%D0%BE%20%D0%90%D0%92%D0%9F%D0%97.pdf/

7. ГОСТ 34.201-89 «Виды, комплектность и обозначение документов при создании автоматизированных систем».

8. ГОСТ 19.201-78 (СТ СЭВ 1627-79) Техническое задание. Требования к содержанию и оформлению».

9. ГОСТ 2.102-68 «Виды и комплектность конструкторских документов».

10. ДСТУ 3973-2000 «Система розроблення та поставлення продукції на виробництво».

11. ГОСТ 2.103-2013 «Единая система конструкторской документации. Стадии разработки».

12. ГОСТ 34.602-89 «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы».

Лабораторна робота №4

«Дослідження способів використання методу інженерії вимог С.Шлейєр і С.Меллора»

Мета роботи: Засвоєння базових знань щодо сутності і застосування методу інженерії вимог С.Шлейєра і С.Меллора.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

6. Ознайомитись з методичною розробкою до лабораторноъ роботи.
7. Ознайомитись з рекомендованною літературою.
8. Дослідити основні принципи методології опису вимог до ІС із використанням підхода Шлейєр і Меллора.
9. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
10. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

4 Назва та мета роботи.

2. Коротко теоретичні відомості

1. Завдання дослідження лабораторної роботи згідно до обраного варіанта.

2. Результати виконаного дослідження.

3. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Інформаційні система та її ПЗ згідно цього методу створюється як сукупність певної множини доменів проблемних областей, кожний з яких являє собою окремий світ зі своїми об’єктами. При цьому кожний такий домен аналізується незалежно від інших.

Продуктом аналізу домену є три моделі [1]:

- інформаційна модель системи (онтологія домену);
- модель станів об’єктів, визначеної у складі інформаційної моделі (або онтології);
- модель процесів, що забезпечують переходи із одного стану об’єкта в інший.

Інформаційна модель домену

На першому етапі треба визначити існуючі об’єкти й виявити зв’язки між ними. Знайдемо об’єкти і дамо їм унікальні імена [1].

Категорії об’єктів, серед яких має сенс проводити пошук:

- реальні предмети світу (стілець, Дніпро і т.п.);
- абстракції фізичних предметів (будинок, літак);
- ролі предметів (для університету – ректор, студент, декан);
- інциденти – події, які впливають на зміну стану об’єктів (повінь, вибори, дефекти тощо);
- взаємодії – це відношення між об’єктами (контракт);
- специфікації – це подання правил, стандартів, критеріїв якості й обмежень на використання системи (правила, стандарти).

Атрибути об’єктів визначаються для кожного об’єкта. Вони являють собою характерні його властивості. Для кожного з атрибутів задаються можливі значення (числовий діапазон, список значень, правила генерації).

Після цього визначається ідентифікатор об’єкта (один або більше атрибутів, які точно вирізняють об’єкт серед інших).

Представлення інформаційної моделі в цьому методі базується на відомій реляційній моделі даних. Об’єкти нумеруються і зображуються у прямокутнику, всередині якого подається номер об’єкта, ім’я об’єкта, а також імена його атрибутів [1].

Об’єкти різних класів можуть брати участь у попарних зв’язках з іншими об’єктами. Розрізняють три види зв’язків:

1. 1:1
2. 1:n
1. m:n

Над стрілкою може вказуватися ім’я (ідентифікатор) зв’язку, а на кінцях зв’язку можуть вказуватися також ролі об’єктів.

Інформаційна модель метода Шлеєр і Меллора базується на відомій моделі Чена (E-R діаграма), що відтворює рис. 1.

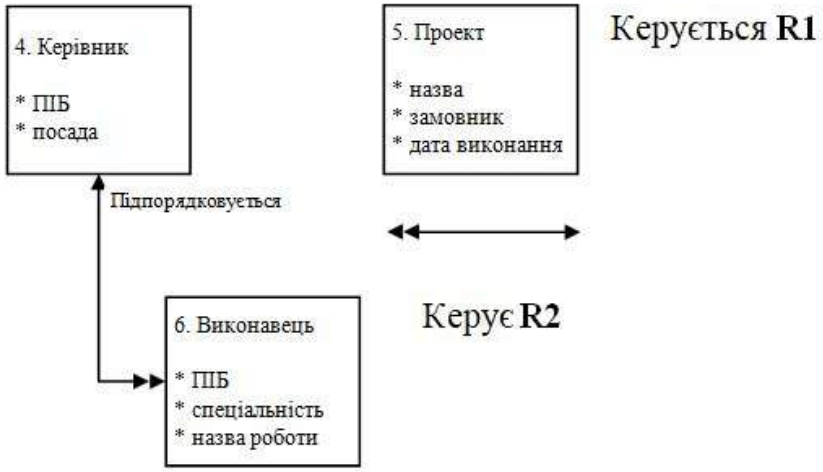


Рис. 1. Приклад інформаційної моделі по методу Шлеєр і Меллора

Тут R1, R2 – ідентифікатори зв’язків. Крім інформаційної моделі як правило будуються відношення успадкування за допомогою відповідних діаграм (рис. 2).

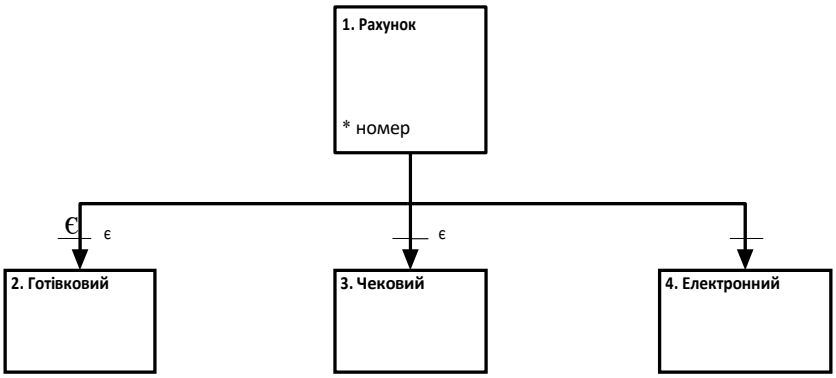


Рис. 2. Приклад відношення спадкування

Модель станів

Дана модель призначена для відображення динаміки змін, що відбуваються в стані кожного з екземплярів класу. Нагадаємо базові поняття моделі динаміки поведінки об’єктів:

- стан об’єкта визначається поточними значеннями його окремих атрибутів;
- стан об’єкта змінюється в результаті подій, що відбулися, або появи певних стимулів;

- стан домену визначається сукупністю станів його об’єктів;
- зміна стану об’єкта супроводжується деякими процесами, які наперед визначені для кожного стану.

Для фіксації динамічних аспектів вимог є дві нотації: діаграма переходів у стани (ДПС) і таблиця переходів у стани (ТПС) [1].

Обидві нотації базуються на автоматі Мура. Для відображення поведінки об’єктів, у вимогах на розробку ПС потрібно:

1) визначити множину станів;
2) визначити множину інцидентів або подій, які спонукують екземпляри класів змінювати свій стан (переходити у новий стан);
3) визначити правила переходу для кожного із зафіксованих станів, що вказують у який стан переходить екземпляр класу;
4) визначити процеси, що виконуються при переході. Графічна нотація ДПС передбачає наступне:

- кожний стан об’єкта має назву й порядковий номер;
- кожній події надається унікальна мітка й назва;
- стан позначається рамкою з номером та назвою стану;
- перехід позначається лінією зі стрілкою;
- використовуються спеціальні системні об’єкти – ***таймери*** як механізми виміру інтервалів часу (атрибути таймеру: унікальний ідентифікатор; інтервал часу, через який подається сигнал про настання певної події; мітка події, яка наступить, коли залишок часу буде дорівнювати 0).

Розглянемо ДПС та ТПС пральної машини (рис. 3 та рис. 4).

ТПС складається з рядків і стовпців, причому кожний з можливих станів об’єкта представляється рядком, а кожна з можливих подій – стовпцем. Клітина ТПС визначає стан у який переходить об’єкт. Якщо порівняти ДПС і ТПС, то перевагою ДПС є наочність і визначення дій, однак ТПС дозволяє зафіксувати всі можливі комбінації стан-подія. За допомогою побудови ТПС забезпечують повноту та несуперечність представлення вимог.

Рис. 3. ДПС автоматичної пральної машини

	П1	П2	П3	П4	П5	П6	П7	П8
С1	2							
С2		3						
С3			4					
С4				5				
С5					6			
С6						7	2	
С7								1(8)

Рис. 4. ТПС автоматичної пральної машини

Все, що подано вище, відноситься до окремих об’єктів, як базових складових архітектури ПС. Для системи в цілому будуються схеми взаємодіючих моделей поведінки об’єктів. Взагалі ПС розглядається як взаємодія об’єктів. Така взаємодія моделюється як обмін повідомленнями між об’єктами системи і зовнішніми об’єктами. Оскільки поведінка об’єктів представляється відповідними ДПС, то поведінка системи в цілому представляється як схема взаємодіючих ДПС.

Модель процесів

Дії є алгоритмами, які виконуються системою, як реакція на зовнішні події. Набір таких дій визначає поведінку системи, її функціональність. Розуміння вимог до системи має на увазі розуміння цих дій, які можуть бути досить складними [2].

Для подолання складності розуміння дій, виконують їх декомпозицію на окремі складові, які називають процесами. Послідовність процесів визначає потік керування. При цьому процеси обмінюються даними, які утворюють потік даних. Для представлення моделі алгоритмів дій системи використовуються діаграми потоків даних дій (ДПДД) [1, 2].

Як джерела даних допускаються:

- атрибути об'єктів (зберігаються у файлах або базах даних);
- системні годинники й таймери;
- дані подій і повідомлення.

Правила побудови ДПДД

Кожному стану з ДПС може відповідати тільки одна ДПДД. Процес на ДПДД зображується у вигляді овалу (усередині дається назва), потоки даних зображуються стрілками (на котрих вказуються ідентифікатори даних, напрямок до овалу – вхідні, а відовалу – вихідні дані), джерела данихзображуються прямокутниками або рамками з відкритими сторонами (рис. 5).

Потоки даних від таймера маркуються номером таймера.

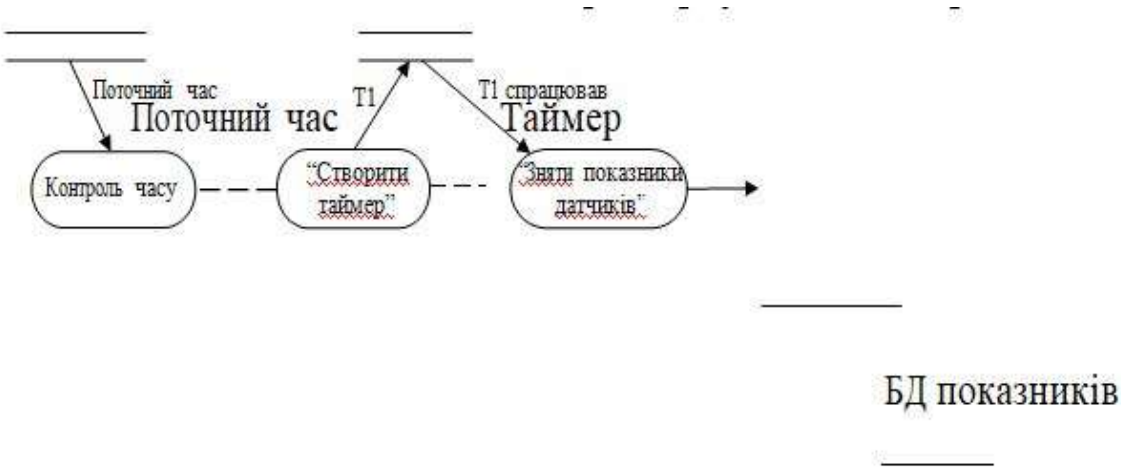


Рис. 5. Приклад діаграми потоків даних дій (ДПДД) для стану "Запис показників датчиків"

Після побудови ДПДД варто побудувати загальну таблицю процесів із наступними колонками:

- ідентифікатор процесу;
- тип процесу (аксесор – доступ до архівів даних, генератор подій, процес-обчислення, перевірка умов);
- повна назва процесу;
- назва стану, у якому визначений процес та назва дії стану; За допомогою цієї таблиці перевіряється:

1. несуперечність назв та ідентифікаторів процесів;
2. повнота подій та процесів;
3. виявляються спільні процеси для різних дій чи станів.

Продукти інженерії вимог за методом Шлеєр і Меллора

Відповіднодо методу,результатами аналізу вимогдо створюваних ПС є наступні продукти [1]:

1. Інформаційна модель системи (онтологія) у формі:
   - діаграми сутність-зв'язок;
   - опис об'єктів з атрибутами та зв'язків між об'єктами.
2. Модель поведінки об'єктів системи у формі:
   - ДПС і ТПС;
   - описи дій для ДПС та описи подій для ДПС та ТПС.
3. Модель процесів для станів об'єктів у вигляді:
   - ДПДД та таблиці процесів станів;
   - описи процесів (природною мовою).

Порядок виконання роботи

В процесі виконання роботи з метою формалізації вимог до ПС в межах заданої предметної галузі побудувати 4 типи діаграм і одну таблицю ТПС, виконавши наступні кроки:

1. Шляхом аналізу предметної області слід виділити суттєві для системи об'єкти (5-6 об'єктів) та побудувати інформаційну модель системи. При цьому можна використати досвід побудови E-R діаграм для заданої предметної області.

2. Віднайти для складних об'єктів відношення спадкування і побудувати для них відповідні діаграми спадкування властивостей.

3. Розглянути модель поведінки для довільного об'єкта системи. Об'єкт має мати не менше 6-7 незалежних станів. Для об'єкта треба побудувати діаграму переходів у стани (ДПС) і відповідну парну до цієї діаграми таблицю переходів у стани (ТПС). При цьому користаємося

діаграмами активності мови UML.

4. Для довільного стану об’єкта із побудованої ДПС побудувати модель процесів у вигляді ДПДД (5-6 процесів/дій та 4- 5 джерел даних). Для побудови цих діаграм можна скористатися діаграмами станів та діаграмами потоків даних. Для ДПДД побудувати загальну таблицю процесів із наступними колонками: ідентифікатор процесу; тип процесу (аксесор, генератор подій, процес-обчислення, перевірка умов); повна назва процесу; назва стану, у якому визначений процес; назва дії стану.

5. Для заданого опису предметної області визначити вимоги до проектованої системи відповідно до методу Шлеєр і Меллора:

1) Діяльність підприємства “Будинок музики”

Підприємство “Будинок музики” займається продажами і ремонтом музичних інструментів (електрогітар, клавішних та ударних інструментів) та апаратури. Крім того підприємство проводить заходи, пов’язані з музичною діяльністю (концерти, джем-сейшени, майстер-класи). Напрямками діяльності є такі: виконання продажів музичних інструментів, ремонт інструментів і обладнання відповідно до замовлень клієнтів, організація музичних заходів, навчання в групах та індивідуально. Покупець оформляє замовлення інструментів у продавця або через Інтернет.

2) Діяльність Інтернет-провайдера

Інтернет-провайдер має автоматизувати обслуговування клієнтів. Основними функціями організації є підключення нових абонентів до мережі Інтернет, здійснення контролю за оплатою ними послуг, оброблення позаштатних ситуацій (включаючи налагодження з’єднання з боку клієнта) та виконання аналізу якості сервісу. Організація поділяється на чотири відділи: абонентський, фінансовий, експлуатації та будівництва мережі. Кожен з відділів для виконання функцій використовує менші структурні одиниці.

3) Діяльність ріелтерської агенції

В агенції, що купує й продає нерухомість, із клієнтами працюють брокери. Агенція має назву, ліцензію, адресу, телефон, директора. На операцію з нерухомістю брокер оформляє накладну, в якій зазначено номер, дату, тип операції (купівля, продаж, оренда), ПІБ брокера, тип застави, атрибути клієнта, адреса об'єкта нерухомості. Нерухомість характеризується державним реєстраційним номером, адресою, вартістю, приналежністю. Клієнтом може бути як фізична, так і юридична особа.

4) Діяльність видавництва

У видавництві, що видає печатну продукцію, клієнти роблять замовлення. Співробітники видавництва характеризуються посадою, фахом і працюють у різних відділах, котрі мають назву та список технічних засобів для друку. Перш ніж оформити замовлення, замовники вивчають прайси, в яких перераховані види й найменування продукції (книжки, журнали, газети, листівки, конверти, плакати), із зазначенням обсягів та термінів робіт, а також вартістю одиниці продукції. Замовлення має список печатної продукції та банківські рахунки виконавця та замовника.

Рекомендована література

6. Бабенко Л.П., Лавріщева К.М. Основи програмної інженерії. Навч. посіб. –К.: Т-во "Знання", КОО, 2001. – 269 с.

7. Брауде Э. Технология разработки программного обеспечения. – СПб.: Питер, 2004. – 655 с.

Лабораторна робота №5

«Дослідження способів використання методу інженерії вимог І.Джекобсона»

Мета роботи є дослідження:

- методології опису вимог до програмної системи (ПС) із використанням підхода Джекобсона, який використовується не тільки для формалізації вимог до ПС, а і для визначення складу об'єктів, якими буде оперувати ПС;

- способу застосовувати діаграми Джекобсона для опису вимог до ПС, що функціонує у визначеній предметній галузі.

.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Дослідити основні принципи методології опису вимог до ІС із використанням підхода Джекобсона.
4. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
5. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

5 Назва та мета роботи.

2. Коротко теоретичні відомості

4. Завдання дослідження лабораторної роботи згідно до обраного варіанта.

5. Результати виконаного дослідження.

6. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Метод Джекобсона – це єдиний метод, що вказує послідовний достатньо формалізований підхід до виявлення об’єктів, суттєвих для даної предметної області [3].

Концепція моделі сценаріїв для збору вимог

Метод Джекобсона базується на варіантах використання системи. На першому кроці складна система декомпозується на більш прості складові [1, 3]. Розробка системи починається з осмислення мети системи, тобто для кого й для чого створюється ПС. Складність загальної мети виражається через окремі складові мети. Складові мети можуть відповідати функціональним і нефункціональним вимогам, а також проектним рішенням.

Функціональні вимоги – це вимоги до функцій системи.

Нефункціональні вимоги – це вимоги, наприклад, до надійності, безпеки, тестованості тощо [2, 4, 5].

Складові мети є джерелом вимог до системи і класифікуються на обов’язкові й бажані. Складові мети можуть перебувати в певних відношеннях (узгодженість, конфлікт, залежність тощо).

Другий крок – визначення носіїв інтересів, яким відповідає кожна складова мети, і можливих сценаріїв. Відбувається послідовна декомпозиція складних проблем:

1) проблеми трансформуються в сукупність складових мети;
2) кожна із складових мети трансформується в сукупність можливих прикладів використання системи;
3) сценарії трансформуються в процесі аналізу в сукупність взаємодіючих об’єктів.

У такий спосіб будується ланцюжок трансформації: проблема – складові мети – сценарії – об’єкти. Проведена трансформація відображається у термінах базових понять проблемної області.

Кожний сценарій запускається користувачем (актором), який є зовнішнім чинником ІС. Кожний сценарій запускає один актор.

Сценарій складається з ряду операцій і має стани й поведінку.

Сценарій – це повний ланцюжок подій, ініційованих актором, і специфікація реакції на них [2, 3, 6].

Сукупність сценаріїв визначає всі можливі шляхи використання системи. Для моделі сценаріїв визначається спеціальна графічна нотація: актор – іконка людини, під якою вказується назва актора; сценарій – овал (усередені овалу – назва сценарію); зв’язки між ними – стрілки. Кожного актора обслуговує своя сукупність сценаріїв (див. рис.1).

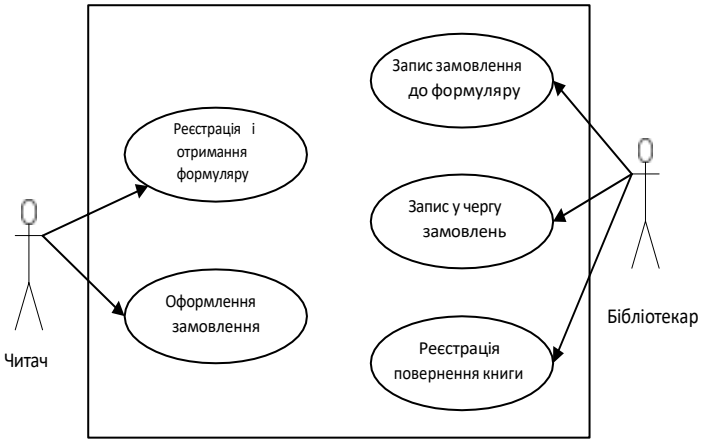


Рис.1. Діаграма сценаріїв бібліотеки

Відношення розширення “extend” вказує на те, що функції одного сценарію є доповненням до функцій іншого. Відношення використання “include” означає, що певний сценарій може бути використаний декількома іншими сценаріями (аналог процедури).

Продуктом першої стадії інженерії вимог – збір вимог по Джекобсону – є модель вимог, що складається із онтології домену, моделі сценаріїв і опису інтерфейсів сценаріїв [1, 2, 4].

Онтологія домену може бути зображена за допомогою нотації моделі сутність-зв'язок (ERD). Модель сценаріїв супроводжується неформальним описом кожного сценарію, що складається з:

- назви, анотації та акторів, які запускають сценарій;
- опису аспектів дій акторів по взаємодії з системою;
- передумови та функції, які виконуються в сценарії;
- нестандартні ситуації й реакція на них;
- умови завершення сценарію і постумови.

Далі сценарій використання трансформується в сценарій поведінки системи шляхом додавання елементів, які пов’язані з конструктивним рішенням цільового продукту. Наприклад, механізми запуску сценарію (пункти меню), механізми вводу даних (ввід пароля), поведінка у випадках виникнення нештатних ситуацій. Опис інтерфейсів дається теж неформально, однак корисно узгоджувати інтерфейси ПС із майбутніми користувачами.

Модель аналізу вимог

Модель вимог піддається аналізу з метою подальшого структурування проблеми, що вирішує дана ПС. Оскільки обрано об'єктну архітектуру, то результатом структурування є об'єкти, взаємодія яких визначає функціональність ПС. Сукупність об'єктів знаходимо шляхом послідовної декомпозиції сценаріїв на об'єкти.

Слід визначити такий набір об'єктів, який дасть можливість ПС бути легко адаптованою у випадку зміни вимог до неї [1, 6].

Аксіома. *Всяка працююча ПС згодом вимагає змін*.

Тому стратегія вибору об'єктів заснована на таких постулатах:

1. зміна вимог неминуча;
2. об'єкт повинен модифікуватися тільки внаслідок зміни відповідних вимог до ПС;
3. об'єкт повинен бути стійким до модифікації;
4. під стійкістю до модифікацій (стабільність ПС) мається на увазі, що зміни будь-якої з вимог спричиняють собою зміну найменшої кількості об'єктів ПС (в ідеалі - одного).

Вищенаведені принципи приводять до того, що простір, у якому функціонує ІС, потрібно структурувати в ***трьох*** вимірах:

1) інформація, що обробляє система (її внутрішній стан);
2) поведінка системи та її презентація (інтерфейси ПС).

Результати досліджень свідчать, що на протязі ЖЦ найбільша кількість змін відбувається в інтерфейсах. Поведінка системи більш консервативна, але також зазнає змін. Характер і структура інформації, яку обробляє ПС, є найбільш стійкими до змін.

Доцільно для кожного з вимірів мати свою сукупність об'єктів, що його відображає. За допомогою такого поділу ми локалізуємо в тексті вимог найбільш стабільні, мобільні й проміжні елементи.

Об'єкт-сутність моделює довгоживучу інформацію, яка зберігається після виконання сценаріїв. Більшість із цих об'єктів виявляються на етапі аналізу проблемної області, але до уваги беруться тільки ті, на які є посилання в сценарії.

Об'єкти-інтерфейси містять функціональності, які залежать безпосередньо від оточення системи. Ці об'єкти визначаються шляхом аналізу опису інтерфейсу сценарію з моделі вимог. За допомогою інтерфейсів актори взаємодіють із кожним зі сценаріїв системи. Завдання об'єкта – трансляція інформації, яку вводить актор, у події, на які реагує система та зворотня трансляція.

Керуючі об'єкти – це об'єкти, які перетворюють інформацію, введену об'єктами інтерфейсу й представлену об'єктами- сутностями, в інформацію, що виводиться інтерфейсними об'єктами. Керуючі об'єкти відповідають функціям обробки інформації і є перетворювачами.

Модель аналізу вимог має відповідну графічну нотацію:

Модель аналізу вимог має відповідну графічну нотацію:

- Об'єкт-сутність - Керуючий об'єкт

- Об'єкт-інтерфейс

Діаграма взаємодії об'єктів в рамках сценарію будується на основі аналізу вимог до цього сценарію. Приклад такої діаграми показано для сценарію обробки замовлень в бібліотеці на рис. 2.

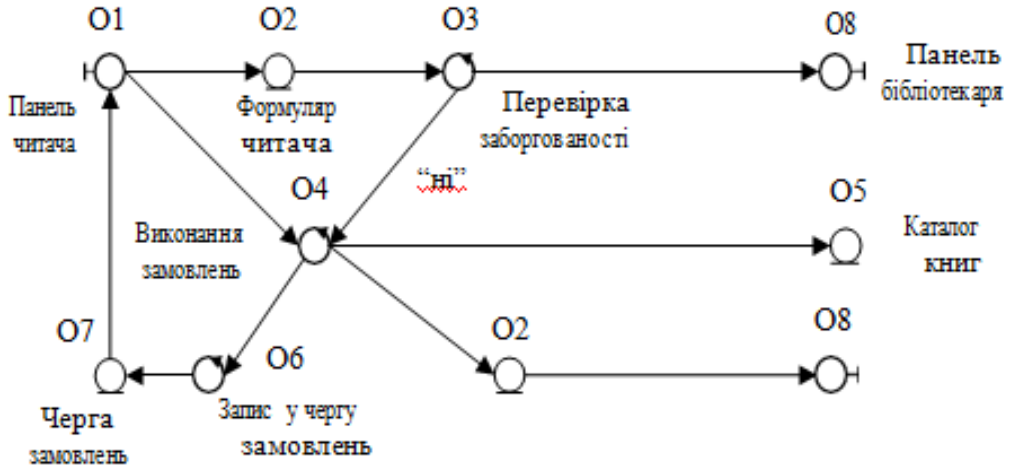


Рис. 2. Модель аналізу вимог: асоціації між об'єктами бібліотечної системи

Таким чином, послідовний підхід до виявлення об'єктів, суттєвих для даної предметної області, складається з таких кроків:

1) Множина можливих об'єктів визначається на етапі побудови інформаційної моделі проблемної галузі (ERD домену).

2) Склад об'єктів уточнюється після побудови комплекту діаграм сценаріїв для моделювання вимог до проектованої ПС (можуть з'явитися нові об'єкти та скоротитися склад об'єктів з п.1, якщо вони не використовуються в сценаріях).

3) Остаточно склад об'єктів визначається після побудови діаграм аналізу вимог для кожного сценарію (можуть з'явитися нові об'єкти та скоротитися склад об'єктів з п.2, якщо вони не використовуються у відповідних діаграмах взаємодії).

Продукти інженерії вимог по методу Джекобсона

1. Онтологія домену (для нотації проблемної області можна скористатися діаграмами Чена або Баркера, тобто ERD).
2. Моделі сценаріїв та неформальний опис сценаріїв і акторів.
3. Детальний опис інтерфейсів, сценаріїв і акторів.
4. Діаграми взаємодії об’єктів в рамках сценаріїв (будуються на основі аналізу вимог).

Подальша деталізація вимог відбувається на етапах ЖЦ ПС, при цьому зберігається трасування вимог (відстеження відповідних об’єктів протягом ЖЦ принаймні на всіх етапах розробки ПС).

Порядок виконання роботи

У процесі виконання роботи з метою формалізації вимог та визначення складу об’єктів в межах заданої предметної галузі, необхідно побудувати 2 типи діаграм:

1) На першому кроці шляхом аналізу предметної області слід виділити суттєві для системи об’єкти (5-6 об’єктів) та побудувати інформаційну модель системи. Для побудови інформаційної моделі використовується, також, діаграма класів.

2) На другому кроці необхідно побудувати повну діаграму сценаріїв для ПС, що створюється. В діаграмі повинно бути 5-6 базових функціональних сценаріїв та 4-5 сценаріїв типу розширення або використання (“extend”, “include”). Для побудови повної розгорнутої діаграми сценаріїв ПС можна скористатися діаграмами варіантів використання (use case diagrams, мова UML).

Після побудови діаграми сценаріїв необхідно скласти неформальний опис всіх сценаріїв і акторів, наявних на діаграмі.

3) На третьому кроці слід побудувати дві діаграми аналізу вимог (діаграми взаємодії об’єктів в рамках сценаріїв) для двох відповідних сценаріїв ПС із діаграми сценаріїв (п.2). На кожній побудованій діаграмі має бути не менше 2-3 об’єктів-інтерфейсів та по 4-5 керуючих об’єктів і об’єктів-сутностей. Для кожного знайденого об’єкта треба виконати детальний опис.

Приклад завдань лабораторної роботи

Для заданого опису предметної області визначити вимоги до проектованої системи відповідно до методу Джекобсона:

1. *Робота готелю*

Обслуговування клієнтів готелю полягає в забезпеченні клієнтів послугами та перевірці якості цих послуг. Послуги забезпечує персонал готелю, а перевірку здійснюють менеджер та адміністратори. Відповідно до створеного дирекцією бізнес-плану готелю надаються такі послуги: забезпечення їжею та напоями як з доставкою у номери готелю, так і у ресторанах та барах готелю; надання номерів зі зручностями та комплексів для відпочинку; забезпечення безпеки мешкання; надання різного роду інформації для гостей. Щомісячно складається бізнес-звіт діяльності готелю.

2. *Діяльність букмекерської контори*

Букмекерська контора приймає ставки на результати наступних спортивних змагань. Оператор-букмекер приймає від клієнта прогнози на результати матчів, а ставки клієнт сплачує касиру в касу закладу, отримуючи квітанцію. Після оприлюднення результатів клієнт (у разі виграшу) може отримати суму свого виграшу в касі, надаючи відповідні квітанції. Раз у тиждень готівку із каси закладу інкасатори вивозять у банк. Персоналу контори суворо забороняється обслуговувати неповнолітніх, що перевіряється зовнішніми інспекторами щоденно.

3. *Робота житлово-комунальної контори*

В кожному місті маються житлово-комунальні контори (ЖКК), котрі здійснюють обслуговування жителів будинків, постачаючи газ, електроенергію та воду. В ЖКК мається декілька відділів (дирекція, бухгалтерія, інспектори по прийому громадян, водії та робітники різних спеціальностей), Жителі будинків сплачують вартість комунальних послуг і отримують квитанції про сплату. На балансі ЖКК, як правило, знаходиться декілька будинків. На виконання робіт жителі будинку подають заявку, а майстер оформляє накладну, де зазначено: дату, адресу, ПІБ замовника, список матеріалів та виконаних робіт із зазначенням вартості.

Рекомендована література

1. Бабенко Л.П., Лавріщева К.М. Основи програмної інженерії. Навч. посіб. –К.: Т-во “Знання”, КОО, 2001. – 269 с.

2. Брауде Э. Технология разработки программного обеспечения. – СПб.: Питер, 2004. – 655 с.

3. Райчев І.Е., Харченко О.Г., Замковий В.В. Принципи проектування відкритих розподілених систем : навч. посіб. –К.: Вид-во Нац. авіац. ун-ту

“НАУ-друк”, 2010. – 240 с.

4. Лавріщева К.М. Програмна інженерія. Підручн. –К.: Академперіодика, 2008. – 320 с.

5. Канер Сэм, Фолк Дж., Нгуен Енг Кен. Тестирование программного обеспечения : Пер. с англ. –К.: Диасофт, 2001. –544с.

6. Соммервилл И. Инженерия программного обеспечения. – М.: Изд. дом Вильямс, 2002. – 624 с.

Лабораторна робота №6

«Дослідження моделей якості програмної системи з використанням метрик стандарту ISO/IEC 9126 (PARTS 2,4)»

Мета роботи:

- дослідження методів побудови моделей якості програмних систем (ПС) відповідно до побудованих моделей якості, що створені згідно рекомендацій стандарту ISO/IEC 9126:

- дослідження характеристик, підхарактеристик та метрик якості стандартів ISO/IEC 9126 (частини 1, 2 та 4) для побудови моделей зовнішньої та експлуатаційної якості ПС, що функціонує у визначеній предметній галузі.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

11.Ознайомитись з методичною розробкою до лабораторноъ роботи.

12. Ознайомитись з рекомендованною літературою.

13. Ознайомитись з діючими стандартами.

14. Дослідити основні принципи методології побудови моделей якості програмних систем з використанням метрик стандарту ISO/IEC 9126 (PARTS 2,4.

15. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..

16. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

6 Назва та мета роботи.

2. Коротко теоретичні відомості

7. Завдання дослідження лабораторної роботи згідно до обраного варіанта.

8. Результати виконаного дослідження.

9. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Визначення якості ПС

Визначення якості сформульоване в ДСТУ 2844-94 «Програмні засоби. Забезпечення якості. Терміни та визначення»:

Якість – це сукупність властивостей ПС, які забезпечують її здатність задовольняти встановлені або передбачувані потреби відповідно до призначення ПС [1, 2, 3].

Для оцінки ПС використовуються в основному два процеси: ***атестація*** та ***сертифікація***. Як правило ***атестація*** проводиться організацією розробником для перевірки того, чи відповідає ПС заданим вимогам. У процесі атестації широко використовується тестування. ***Сертифікація*** проводиться третьою стороною, а саме спеціальним органом, ліцензованим державою для таких дій.

Сертифікація – це оцінка відповідності властивостей ПС вимогам до неї. *Сертифікаційні випробування* виконують лабораторії, що призначаються органом сертифікації [9].

Оцінювання якості ПС

Атрибути ПС, які характеризують якість, виміряються за допомогою ***метрик якості.***

В ієрархічній моделі якості ПС, яка складається із множини характеристик, введені такі позначення: C_i – характеристики, S_i– підхарактеристики, A_i – атрибути якості ПС.

Питання побудови моделей якості та оцінювання якості розглянуті у стандартах ISO/IEC 9126 (частини 1, 2, 3 та 4): модель якості – частина 1, зовнішні метрики – частина 2, внутрішні метрики – частина 3, експлуатаційні метрики – частина 4.

Поняття якості програмного забезпечення (ПЗ) відповідає уявленню про те, що програма досить успішно справляється з всіма покладеними на неї задачами та не приносить проблем ні кінцевим користувачам, ні їхньому начальству, ні службі підтримки, ні фахівцям по продажах.

Якість програмного забезпечення визначається в стандарті ISO 9126 як вся сукупність його характеристик, що належать до можливості задовольняти висловлені або ті, що маються на увазі потреби всіх зацікавлених осіб

Розрізняються поняття **внутрішньої якості**, пов'язаного з характеристиками ПЗ самого по собі, без урахування його поведінки; **зовнішньої якості**, що характеризує ПЗ із точки зору його поведінки; і якості ПЗ **при використанні** у різних контекстах - тої якості, що

відчувається користувачами при конкретних сценаріях роботи ПЗ. Для всіх цих аспектів якості уведені метрики, що дозволяють оцінити їх. Крім того, для створення добротного ПЗ істотно якість технологічних процесів його розробки. Взаємини між цими аспектами якості за схемою, прийнятою ISO 9126, показане на рис. 1.

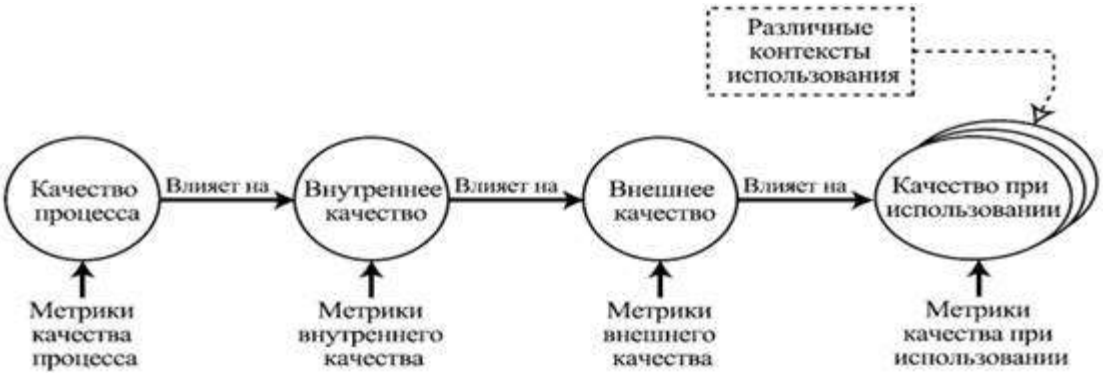


Рис. 1. Основні аспекти якості ПЗ за ISO 9126

Системи керування якістю – вимоги до ПЗ та моделі для забезпечення якості при проектуванні, розробці, комерціалізації, установці та обслуговуванні

Стандарт ISO 9126 визначає загальні правила забезпечення якості результатів у всіх процесах життєвого циклу.

Цей стандарт виділяє наступні процеси:

- управління якістю;
- управління ресурсами;
- розвиток системи управління;
- дослідження ринку;
- проектування продуктів;
- придбання;
- виробництво;
- надання послуг;
- захист продуктів;
- оцінка потреб замовників;
- підтримка комунікацій із замовниками.;
- підтримка внутрішніх комунікацій;
- управління документацією;
- ведення записів про діяльність.;
- планування;
- навчання персоналу;
- внутрішні аудити;

- оцінки управління;
- моніторинг і виміри;
- управління невідповідностями;
- постійне вдосконалювання;
- управління та розвиток системи в цілому.

Для кожного процесу потрібно мати плани розвитку процесу, що складаються як мінімум з наступних розділів:

- проектування процессу;
- документування процессу;
- реалізація процессу;
- підтримка процессу;
- моніторинг процессу;
- управління процессом;
- удосконалення процессу.

Крім підтримки й розвитку системи процесів, націлених на задоволення потреб замовників і користувачів, ISO 9001 вимагає:

- визначити, документувати та розвивати власну систему якості на основі вимірних показників;
- використовувати цю систему якості як засіб управління процесами, націлюючи їх на більше задоволення потреб замовників, плануючи й постійно відслідковуючи якість результатів всіх видів діяльності, у тому числі й самого управління;
- забезпечити використання якісних ресурсів, якісного (компетентного, професійного) персоналу, якісної інфраструктури і якісного оточення;
- постійно контролювати дотримання вимог до якості на практиці, у всіх процесах проектування, виробництва, надання послуг і при придбаннях;
- передбачити процес усунення дефектів, визначити й контролювати якість результатів цього процессу.

Визначення характеристик і атрибутів за стандартом ISO 9126:2001

Функціональність (functionality).

Здатність ПЗ в певних умовах вирішувати задачі, потрібні користувачам. Визначає, що саме робить ПЗ, які задачі воно вирішує

Функціональна придатність (suitability).

Здатність вирішувати потрібний набір задач.

Точність (accuracy).

Здатність видавати потрібні результати.

Здатність до взаємодії (interoperability).

Здатність взаємодіяти з потрібним набором інших систем.

Відповідність стандартам і правилам (compliance).

Відповідність ПЗ наявних індустріальних стандартах, нормативним і законодавчим актам, іншим регулюючим нормам.

Захищеність (security).

Здатність запобігати неавторизированому, тобто без вказівки особи, що намагається його здійснити, і недозволеному доступу до даних і програм.

Надійність (reliability).

Здатність ПЗ підтримувати визначену працездатність у заданих умовах.

Зрілість, завершеність (maturity).

Величина, зворотна частоті відмов ПЗ. Звичайно виміряється середнім часом роботи без збоїв і величиною, зворотною імовірності виникнення відмови за даний період часу.

Стійкість до відмов (fault tolerance).

Здатність підтримувати заданий рівень працездатності при відмовах і порушеннях правил взаємодії з оточенням.

Здатність до відновлення (recoverability).

Здатність відновлювати визначений рівень працездатності й цілісність даних після відмови, необхідні для цього час і ресурси.

Відповідність стандартам *надійності* ***(reliability compliance).***

Атрибут доданий в 2001 році.

Зручність використання (usability) або практичність.

Здатність ПЗ бути зручним у навчанні та використанні, а також привабливим для користувачів.

Зрозумілість (understandability).

Показник, зворотний до зусиль, які затрачаються користувачами на сприйняття основних понять ПЗ та усвідомлення їх застосовності для розв'язання своїх задач.

Зручність навчання (learnability).

Показник, зворотний зусиллям, затрачуваним користувачами на навчання роботі з ПЗ.

Зручність роботи (operability).

Показник, зворотний зусиллям, що вживається користувачами для розв'язання своїх задач за допомогою ПЗ.

Привабливість (attractiveness).

Здатність ПЗ бути привабливим для користувачів. Атрибут доданий в 2001 році.

Відповідність стандартам зручності використання (usability compliance).

Атрибут доданий в 2001 році.

Продуктивність (efficiency) або ефективність.

Здатність ПЗ при заданих умовах забезпечувати необхідну працездатність стосовно виділюваного для цього ресурсам. Можна визначити її і як відношення одержуваних за допомогою ПЗ результатів до затрачуваних на це ресурсів усіх типів.

Часова ефективність (time behaviour).

Здатність ПЗ видавати очікувані результати, а також забезпечувати передачу необхідного об'єму даних за відведений час.

Ефективність використання ресурсів (resource utilisation).

Здатність вирішувати потрібні задачі з використанням визначених об'ємів ресурсів визначених видів. Маються на увазі такі ресурси, як оперативна й довгострокова пам'ять, мережні з'єднання, пристрої вводу та виводу та ін.

Відповідність стандартам продуктивності (efficiency compliance).

Атрибут доданий в 2001 році.

Зручність супроводу (maintainability).

Зручність проведення всіх видів діяльності, пов'язаних із супроводом программ.

Аналізованість (analyzability) або зручність проведення аналізу.

Зручність проведення аналізу помилок, дефектів і недоліків, а також зручність аналізу необхідності змін і їх можливих наслідків.

Зручність внесення змін (changeability).

Показник, зворотний трудозатратам на виконання необхідних змін.

Стабільність (stability).

Показник, зворотний ризику виникнення несподіваних ефектів при внесенні необхідних змін.

Зручність перевірки (testability).

Показник, зворотний трудозатратам на проведення тестування і інших видів перевірки того, що внесені зміни привели до потрібних результатів.

Відповідність стандартам *зручності супроводу* ***(maintainability compliance).***

Атрибут доданий в 2001 році.

Переносимість (portability).

Здатність ПЗ зберігати працездатність при перенесенні з одного оточення в інше, включаючи організаційні, апаратні й програмні аспекти оточення.

Іноді ця характеристика називається у нашій літературі мобільністю. Однак термін "мобільність" варто зарезервувати для перекладу "mobility" - здатності ПЗ й комп'ютерної системи в цілому зберігати працездатність при її фізичному переміщенні в просторі.

Адаптованість (adaptability).

Здатність ПЗ пристосовуватися різним оточенням без проведення для цього дій, крім заздалегідь передбачених.

Зручність установки (installability).

Здатність ПЗ бути встановленим або розгорнутим у визначеному оточенні.

Здатність до співіснування (coexistence).

Здатність ПЗ співіснувати з іншими програмами у загальному оточенні, ділячи з ними ресурси.

Зручність заміни (replaceability) іншого ПЗ даним.

Можливість застосування даного ПЗ замість інших програмних систем для вирішення тих же задач у певному оточенні.

Відповідність стандартам переносимості (portability compliance).

Атрибут доданий в 2001 році.

Перераховані атрибути належать до внутрішньої та зовнішньої якості ПЗ згідно ISO 9126. Для опису якості ПО при використанні стандарт ISO 9126-4 пропонує інший, більш вузький набір характеристик

Ефективність (effectiveness).

Здатність ПЗ надавати користувачам можливість вирішувати їхньої задачі з необхідною точністю при використанні в заданому контексті.

Продуктивність (productivity).

Здатність ПЗ надавати користувачам визначені результати в рамках очікуваних витрат ресурсів.

Безпека (safety).

Здатність ПЗ забезпечувати необхідно низький рівень ризику завдання втрат життю й здоров'ю людей, бізнесу, власності або навколишньому середовищу.

Задоволення користувачів (satisfaction).

Здатність ПЗ приносити задоволення користувачам при використанні в заданому контексті.

Крім перерахованих характеристик і атрибутів якості, стандарт ISO 9126:2001 визначає набори метрик для оцінки кожного атрибута.

Перераховані характеристики та атрибути якості ПЗ дозволяють систематично описувати вимоги до нього, визначаючи, які властивості ПЗ за даною характеристикою хочуть бачити зацікавлені сторони.

Наведені атрибути якості закріплені в стандартах, але це не означає, що вони цілком вичерпують поняття якості ПЗ. Так, у стандарті ISO 9126 відсутні характеристики, пов'язані з **мобільністю ПЗ (mobility)**, тобто здатністю програми працювати при фізичних переміщеннях машини, на якій вона працює. Замість надійності багато дослідників воліють розглядати більш загальне поняття **добротності (dependability)**, що описує здатність ПЗ підтримувати визначені показники якості за основними характеристиками (функціональності, продуктивності, зручності використання) із заданими ймовірностями виходу за їх рамки та визначеним максимальним збитком від можливих порушень. Крім того, активно досліджуються поняття зручності використання, безпеці й захищеності ПЗ, - вони здаються більшості фахівців набагато більш складними, ніж це описується даним стандартом

Моделі якості ПС

Модель якості ПС представляє множину взаємозалежних характеристик, які утворюють базис для специфікації вимог до якості і оцінки якості ПС. Як еталон слугує стандарт ISO/IEC9126 (parts 1-4). Зовнішні й внутрішні метрики служать для вимірювання атрибутів моделі. Модель якості показана на рис. 2.

Функціональність (functionality) – властивість ПС виконувати функції у відповідності встановленим і очікуваним потребам при використанні у вказаних умовах.

Надійність (reliability) – властивість ПС зберігати рівень функціонування при роботі в зазначених умовах.

Зручність використання (useability) – властивість ПС бути зрозумілою, освоюваною, зручною і привабливою для користувачів, при використанні в зазначених умовах.

Ефективність (efficiency) – властивість ПС забезпечувати раціональне використання виділених ресурсів при роботі у встановлених умовах.

Супроводжуваність (maintainability) – властивість ПС забезпечувати можливість її ефективної модифікації. Модифікація може включати корегування, вдосконалення або адаптацію ПС до змін середовища, вимог або функціональних специфікацій.

Переносимість (portability) – властивість ПС бути перенесеною з одного середовища до іншого.

Для кожної підхарактеристики якості існує декілька різних метрик якості.Тому в моделі якості запровадимо атрибути. Кожний атрибут буде вимірюватися за допомогою відповідної метрики.

Рис. 2. Модель якості ПС

Таким чином,***загальна формула моделі якості*** буде виглядати так:

$$Q = \left\{ C_i \left\{ S_{ij} \left\{ A_{ijk} \left(M_{ijk}, W_{ijk} \right) \right\}_{k=1}^{K_{ij}} \right\}_{j=1}^{J_i} \right\}_{i=1}^{I} \qquad (1)$$

У формулі (1): Q – множина взаємозалежних характеристик (властивостей) якості, C_i – i-та характеристика якості, S_{ij} – j-та підхарактеристика i-ї характеристики якості, A_{ijk} – k-й атрибут j-ї підхарактеристики i-ї характеристики якості, M_{ijk} – метрика k-го атрибута j-ї підхарактеристики i-ї характеристики якості, W_{ijk} – коефіцієнт ваги (значимості) k-го атрибута j-ї підхарактеристики i-ї характеристики якості; i, j, k – індекси, I – загальна кількість характеристик якості в моделі якості $(2 \le I \le 6)$, J_i – загальна кількість підхарактеристик i-ї характеристики якості, K_{ij} – загальна кількість атрибутів j-ї підхарактеристики i-ї характеристики якості.

Після визначення всіх елементів моделі (1) можна перейти до випробувань, в ході яких обчислюються фактичні значення атрибутів підхарактеристик якості. Ці значення порівнюються із обмеженнями, заданими у вимогах.

Якщо отримані фактичні значення показників якості відповідають вимогам, то подальшу оцінку можна провести, використовуючи інтегральний показник якості. Інтегральний показник можна застосувати для вибору кращого ПЗ із декількох конкуруючих в даній галузі при умові погодження конфліктуючих атрибутів якості. Але, оскільки в моделі є показники з різними метриками, такими, як неперервні числові, бальні, якісні та інші, необхідно попередньо провести узгодження та нормування метрик, що можна виконати, наприклад, шляхом уведення шкал для якісних та категорійних критеріїв і заданням вагових множників.

Модель експлуатаційної якості

На відміну від моделі зовнішньої і внутрішньої якості, для опису експлуатаційної якості служить однорівнева модель, яка розподіляє атрибути якості по чотирьом характеристикам.

Характеристики експлуатаційної якості:

- *результативність (effectiveness)* – ступінь у якій користувач досягає заданих цілей по точності й повноті вирішення задач у контексті використання ПС;
- *продуктивність (productivity)* – ступінь у якій витрачаються ресурси на досягнення користувачем заданої ефективності у встановленому контексті використання ПС;
- *безпека (safety)* – рівень ризику завдання матеріальних і моральних збитків, тобто шкоди здоров'ю людей, бізнесу, майну, навколишньому середовищу при використанні ПС у встановленому контексті її використання;
- *задоволеність (satisfaction)* – ступінь у якій користувач задоволений ПС у певному контексті її використання.

Порядок виконання роботи

Вважається, що ПС створюється для функціонування в межах попередніх лабораторних роботах.

У процесі виконання роботи необхідно побудувати дві моделі якості для оцінювання рівня якості ПС, а саме: модель зовнішньої якості ПС і модель якості у використанні (модель експлуатаційної якості ПС.

Для цього слід виконати такі кроки:

1). В процесі виконання роботи студенти з метою побудови моделі зовнішньої якості ПС користуються рекомендаціями стандарту ISO/IEC 9126-1. Необхідно обрати не менше 3-5 характеристик якості і для них загалом не менше 9-10 підхарактеристик. Кожну підхарактеристику будемо оцінювати за допомогою одного атрибута якості.

2) Необхідно провести відображення функціональних та нефукціональних вимог до ПС на підхарактеристики та атрибути якості моделі із рис.3.1. Після цього потрібно обгрунтувати обрані властивості якості, що увійшли у модель, тобто пояснити чому ті, а не інші атрибути цікавлять нас більше. На наступному кроці потрібно знайти попарно конфліктуючі між собою властивості якості (не менше двох атрибутів). Наприклад, здатність до співіснування та захищеність і т.ін. Далі студенти обирають вагові коефіцієнти для атрибутів, підхарактеристик та характеристик якості. Побудована модель якості має відповідати формулі (1).

3) Користуючись стандартом ISO/IEC 9126-4, побудувати експлуатаційну модель якості ПС, що функціонує в заданій предметній галузі. Необхідно обрати не менше чотирьох характеристик якості у використанні та відповідні метрики для обчислення досягнутого рівня експлуатаційної якості. Слід визначити яким чином характеристики та атрибути зовнішньої якості відображаються на характеристики експлуатаційної якості.

Приклади завдань

Для заданої предметної галузі побудувати моделі якості програмної системи, наприклад:

1. Робота метеорологічної станції

Метеорологічна станція забезпечує автоматичний контроль наступних погодних характеристик: швидкість та напрям вітру, температура повітря, атмосферний тиск, вологість повітря. Інформаційна система станції дає можливість визначати коефіцієнт різкості погодних умов, зміну температури та тиску. Станція має приміщення для устаткування та установки приладів, обслуговуючий персонал та комп'ютерні засоби. На основі вимірів погодних параметрів складається прогноз погоди, який курьер станції доставляє на радіо та телебачення.

2. Діяльність підприємства по розробці програмних продуктів

Приватні підприємства, що розробляють програмні продукти (ПП), мають декілька відділів, де працюють співробітники (дирекція, бухгалтерія, відділ розробки ПП, відділ тестування ПП, відділ супроводження та ін.). Програмний продукт поставляється замовнику і має назву, список розробників, вартість, документацію, дистрибутив, правила використання. Замовниками можуть бути як фізичні, так і юридичні особи. Кожний замовник може замовити декілька ПП, на кожний з яких він отримує ліцензію, в якій вказано назву продукту, дату продажу, вартість, терміни апгрейдів.

3. Робота поліклініки

В кожному районі міста мається поліклініка, яка обслуговує пацієнтів. Пацієнти записуються на прийом до лікарів-спеціалістів, а також до медсестер на процедури на певну дату і час у деякі кабінети. Пацієнти характеризуються ПІБ, номером медичної картки, номером медичної страховки, адресою. Лікарі назначають пацієнтам лікування у вигляді ліків та процедур. Ліки мають назву, номер ліцензії МОЗ, ціну, правила вживання, термін придатності. Процедури мають назву, список необхідних ліків, тривалість.

Рекомендована література

1. Лавріщева К.М. Програмна інженерія. Підручн. –К.: Академперіодика, 2008. – 320 с.

2. Основы инженерии качества программных систем / Ф.И.Андон, Г.И.Коваль., Т.М.Коротун, Е.М.Лаврищева, В.Ю.Суслов. –К.: Академпериодика, 2007. –672 с.

3. Липаев В.В. Выбор и оценивание характеристик качества программных средств. Методы и стандарты. –М.: СИНТЕГ, 2001. – 228 с.

4. Основы инженерии качества программных систем / Ф.И.Андон, Г.И.Коваль., Т.М.Коротун, Е.М.Лаврищева, В.Ю.Суслов. –К.: Академпериодика, 2007. –672 с.

Лабораторна робота №7

«Дослідження принципів оцінювання рівня якості програмної системи з використанням метрик стандарту ISO/IEC 9126 (PARTS 2,4)»

Мета роботи є:

- дослідження методів оцінки рівня якості ПП відповідно до побудованих моделей якості, що створені згідно рекомендацій стандарту ISO/IEC 9126;

- застосування характеристик, підхарактеристик та метрик якості стандартів ISO/IEC 9126 (частини 1, 2 та 4) для побудови моделей зовнішньої та експлуатаційної якості ПС, що функціонує у визначеній предметній галузі.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Ознайомитись з діючими стандартами.
4. Дослідити основні принципи методології побудови моделей якості програмних систем з використанням метрик стандарту ISO/IEC 9126 (частини 1, 2 та 4).
5. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
6. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

1. Назва та мета роботи.
2. Коротко теоретичні відомості
3. Завдання дослідження лабораторної роботи згідно до обраного варіанта.
4. Результати виконаного дослідження.
5. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Оцінювання якості програмних систем

Атрибути ПС, які характеризують якість, вимірюються за допомогою ***метрик якості.***

Метрика – це комбінація конкретного *методу вимірювання* атрибута сутності й *шкали вимірювання* [2, 3].

Метод вимірювання визначає спосіб одержання значень атрибута якості певної сутності. *Шкала* використовується для структурування отриманих значень. ***Метрика*** визначає *міру* атрибута, тобто змінну, котрій присвоюється значення в результаті вимірювання. Наприклад, сутність – це звіт про виявлення дефектів у ПС, атрибут – список дефектів, метод вимірювання – підрахувати кількість дефектів у списку, шкала – цілі числа більше 0, міра атрибута – загальне число дефектів, ім'я метрики (однойменне мірі) – загальне число дефектів [7].

Міра атрибута є безпосередньою, якщо вона не залежить від мір інших атрибутів, або побічною, якщо утворюється на основі мір інших атрибутів. Питання, пов'язані з побудовою моделей якості докладно розглянуті в попередніх лабораторних роботах.

Оскільки метрика якості ПС – це модель вимірювання атрибута, то метрики служать індикатором одного або декількох атрибутів. Метрика називається ***базовою***, якщо в її основі лежить елементарний метод вимірювання атрибута. Стандарт ISO/IEC 9126-2 рекомендує застосовувати 5 видів шкал виміру значень:

1) *номінальна шкала* (класифікаційна шкала);

2) *порядкова шкала* дозволяє впорядковувати властивості по зростанню/зменшенню шляхом порівняння із базовим значенням;

3) *інтервальна шкала* – відзначає дистанцію між властивостями об'єкта (використовується в арифметичних операціях та операціях порівняння);

4) *відносна шкала* – значення розрізняються відносно обраної одиниці вимірювання(вважається найбільш пріорітетною шкалою);

5) *абсолютна шкала* є спеціальним випадком відносної шкали, де вказується абсолютне значення величини.

Обчисленні значення метрики самі по собі не несуть інформації про рівень задоволення вимог до якості. Для цих цілей шкалу, як правило, ділять на області (ранги), які відповідають різним ступеням задоволеності вимог. Наприклад, можна розподілити шкалу на 4 категорії, як показано на рис. 1.

Стосовно об’єкта вимірювання міри й метрики підрозділяються на *зовнішні*, *внутрішні* й *метрики використання*.

Зовнішні метрики використовують міри ПП, який працює на комп’ютері. Ці міри отримуються в результаті вимірювання поведінки ПП у ході тестування й функціонування. Відповідні метрики розроблюються та використовуються для демонстрації якості ПП, а також для підтвердження того, що програмний продукт задовольняє зовнішнім вимогам до якості.

Рис. 1. Рівні ранжування метрик

Приклади зовнішніх метрик для характеристики надійність – число усунутих дефектів, середній час між відмовами.

Внутрішні метрики дають можливість оцінити якість ПС безпосередньо по її властивостях (об’єм коду, число цикломатичності програми тощо). Внутрішні метрики відображають якість проміжного й кінцевого програмного продукту по тим характеристикам, які визначені в моделі. Ці метрики використовуються для верифікації того, що проміжний та кінцевий ПП задовольняють вимогам до внутрішньої якості. Розробка внутрішніх метрик ґрунтується на виконанні вимірювань статичних атрибутів, які визначаються й оцінюються по тексту вихідного коду, а також по графічому представленню потоків керування, чи по документації на ПС.

Приклади внутрішніх метрик для характеристики надійності:

– число помилок знайдених при інспекції коду, число усунутих дефектів при інспекції [3].

Метрики якості у використанні (експлуатаційні метрики). Ці метрики вимірюють ступінь, у якій ПП задовольняє потреби користувачів в ефективності, продуктивному й безпечному вирішенні завдань і оцінюють видимі результати експлуатації ПС.

Приклади – повнота досягнення цілей користувачів, точність результатів, продуктивність праці, думка користувачів.

Вищевикладені питання розглянуті у стандарті ISO/IEC 9126 (parts 1-4): модель якості – part 1; зовнішні метрики – part 2; внутрішні метрики – part 3; експлуатаційні метрики – part 4.

Оцінювання рівня якості ПС

З*агальна формула моделі якост*і приведена в попередніх лабораторних роботах. Таким чином, інтегральний (узагальнений) рівень якості ПС U_q можна обчислити як середньозважений показник [4]:

$$\begin{cases} U_{ij} = \sum_{i=1}^{I} W_i \sum_{j=1}^{J_i} W_{ij} \sum_{k=1}^{K_{ij}} W_{ijk} Q_{ijk} \\ Q = \left\{ C_i \left\{ S_{ij} \left\{ A_{ijk} \left(M_{ijk}, W_{ijk} \right) \right\}_{k=1}^{K_{ij}} \right\}_{j=1}^{J_i} \right\}_{i=1}^{I} \end{cases} . \quad (1)$$

У формулі (1): W_i – коефіцієнт ваги *i*-ї характеристики якості, W_{ij} – коефіцієнт ваги *j*-ї підхарактеристики *i*-ї характеристики якості, W_{ijk} – коефіцієнт ваги *k*-го атрибута *j*-ї підхарактеристики *i*-ї характеристики якості, Q_{ijk} – рівень якості *k*-го атрибута *j*-ї підхарактеристики *i*-ї характеристики якості, який обчслюється по наступній формулі:

$$Q_{ijk} = \frac{P_{ijk}}{PB_{ijk}} . \quad (2)$$

У формулі (2): P_{ijk} – обчислений рівень якості атрибута, PB_{ijk} – базовий рівень якості згідно обраної метрики (максимально можливе значення міри атрибута).

Приведемо приклад обчислення рівня якості *атрибута*, який вимірюється відповідно до метрики *completeness (закінченість, завершеність)* підхарактеристики *suitability* (*функціональна повнота*) характеристики *функціональність*. Це буде перший атрибут першої підхарактеристики першої характеристики якості моделі, тобто: Q_{111}. Міра

цього атрибута вимірюється відповідно до метрики *X*. Рівень якості цього атрибута обчислюється по формулі:

$$Q_{111} = X = 1 - \frac{A}{B}. \quad (3)$$

У формулі (3): *A* – кількість нереалізованих функцій; *B* – кількість функцій, котрі мають бути реалізовані відповідно до специфікації описів ПП. Тут $0 \le X \le 1$. Природно, що чим X ближче до 1, тим краще.

Порядок виконання роботи

Вважається, що ПС створена для функціонування в межах заданої предметної області відповідно до вимог, визначених в попередніх лабораторних роботах. ПС має повністю відповідати переліку вимог, визначених в попередніх лабораторних роботах.

В процесі виконання попередніх лабораторних робіт були побудовані дві моделі якості для оцінювання рівня якості програмної системи. Це модель зовнішньої якості і модель якості у використанні (модель експлуатаційної якості). Рівень якості ПС слід розраховувати так:

1) Рівень якості, досягнутий кожним атрибутом із формули (1), будемо вимірювати за допомогою зовнішніх метрик, які обираються із стандарту ISO/IEC 9126-2 для відповідних підхарактеристик. Для значень кожної метрики треба побудувати відповідну шкалу ранжування (рис. 1). У цій шкалі мають бути відображені відмінні, добрі, задовільні та незадовільні множини числових значень у балах.

2) Якщо всі конфлікти у моделі узгоджені та виміряні значення показників якості відповідають вимогам, то подальшу оцінку можна провести, використовуючи інтегральний показник якості. Інтегральний рівень якості ПС слід обчислювати по формулі (1). Міри якості атрибутів слід обчислювати, зважаючи на формули (2) і (3). Однак маємо на увазі, що формули метрик для обчислення відповідних мір обираємо із стандарту ISO/IEC 9126-2 для кожного атрибута якості ПС. В результаті виконаних дій слід отримати загальну шкалу ранжування, яку можна застосувати для оцінювання інтегрального рівня якості ПС. У цій шкалі мають бути відображені сукупні множини відмінних, добрих, задовільних та незадовільних числових значень рівня якості ПС із зазначенням граничних балів.

3) Користуючись стандартом ISO/IEC 9126-4, в попередніх лабораторних роботах було побудовано експлуатаційну модель якості ПС, що функціонує в заданому предметному середовищі. Слід обрати не менше чотирьох характеристик якості у використанні та відповідні

метрики для обчислення досягнутого рівня експлуатаційної якості. Необхідно також визначити, як (яким чином) характеристики та атрибути зовнішньої якості відображаються на характеристики експлуатаційної якості ПС. Інтегральний рівень експлуатаційної якості ПС обчислюється так само, як він обчислювався для зовнішньої якості.

Завдання

Орієнтовний перелік предметних галузей проведення дослідження з метою розрахунку рівня якості програмних систем:

1. Підготовка та захист дипломних проектів

Дипломники виконують дипломні проекти на випускаючій кафедрі по конкретній спеціальності. Дипломник має керівників по основній частині проекту та по охороні праці. Кожний керівник може керувати декількома дипломниками. Випускаюча кафедра характеризується назвою, номером і назвою спеціальності. Дипломник характеризується ПІБ, № групи, темою диплому. Керівник характеризується ПІБ, вчена ступінь, посада. Дипломні проекти захищаються у Державній екзаменаційній комісії.

2. Діяльність будівельної організації

Будівельна організація будує різні будинки (житлові, котеджі, офіси та ін.) на замовлення. На будівництво будинку оформлюється наряд, де вказано номер, постанову міськради, адресу будівельного майданчика, вартість, дату початку й дату закінчення робіт. В будівництві приймають участь декілька бригад робітників різних спеціальностей, які використовують різні матеріали. Будівля характеризується типом, адресою майданчика, виконробом, датою здачі. Одна бригада може працювати на декількох будовах.

3. Діяльність авіакомпанії

Авіакомпанія виконує пасажиро- та вантажоперевезення й характеризується назвою, номером ліцензії, адресою, аеропортом базування. Перевезення виконуються повітряними судами (ПС) певними рейсами. Рейс має назву, номер, аеропорт відправлення та аеропорт прибуття, кількість пасажирів і/або вантажу, дату, час. На перевезення вантажів виписується накладна, із зазначенням вартості перевезення одиниці вантажу. В авіакомпанії на різних посадах працюють співробітники, які обслуговують ПС та рейси і характеризуються ПІБ, спеціальністю та посадою.

Приклад виконання дослідження

Теоретичні відомості

Розроблено сучасний програмний засіб для визначення якості програмного забезпечення (ПЗ) методами метричного аналізу, що дає змогу за допомогою показників якості розрахувати відповідні метрики і визначити значення комплексного показника якості програмного продукту. З'ясовано особливості процесу оцінювання якості ПЗ, тобто проаналізовано поняття якість програмного продукту як предмет стандартизації, а також рівні подання моделі якості ПЗ, що дало змогу встановити можливість її підвищення шляхом формування відповідних вимог до критеріїв оцінювання якості, вдосконалення моделей метричного аналізу його якості та методів кількісного її вимірювання на всіх етапах реалізації програмного проекту. Встановлено особливості використання метричного аналізу для визначення якості ПЗ, згідно з якими існує відсутність єдиних стандартів на метрики, тому кожен постачальник вимірювальної системи пропонує власні методи оцінювання якості ПЗ і відповідні метрики. Також є складним завдання інтерпретації значень метрик, позаяк для більшості користувачів як метрики, так і їх значення не зовсім є зрозумілими та інформативними. Виявлено, що основними параметрами при виборі варіанту реалізації ПЗ є його вартість та тривалість процесу розроблення й репутація фірми-проектанта, але рішення, прийняті на підставі цих параметрів, не завжди гарантують належну якість ПЗ. Ключові слова: інформаційні технології; програмний проект; стандарти якості; вимоги до програмного забезпечення; специфікація вимог; критерії оцінювання якості; показники якості програмного забезпечення; комплексний показник якості.

Вступ. Якість програмного забезпечення (ПЗ) є основною його характеристикою в різних сферах використання інформаційних технологій (Pleskach, Zatonatska, 2011), яка вказує на ступінь його відповідності встановленим вимогам (ISO 9001, 2008). Зазвичай, такі вимоги трактують по-різному, що породжує декілька незалежних визначень цього терміна. Здебільшого, під якістю ПЗ розуміють набір властивостей програмного продукту, що характеризують його здатність задовольнити встановлені або передбачувані потреби замовника, які він висловив у вигляді користувацьких вимог до нього на початкових етапах його розроблення (Pomorova & Hovorushchenko, 2013a, 2013b). Стандарт ISO/IEC 9126 (1991) регламентує зовнішні й внутрішні характеристики якості ПЗ. Якщо зовнішні характеристики відображають вимоги до функціонування ПЗ, то внутрішні характеристики використовують для підготовки плану досягнення необхідних значень зовнішніх характеристик його якості

(ISO/IEC TR 9126-2, 9126-3, 2003). Як зовнішні, так і внутрішні характеристики якості відображають властивості самого ПЗ, а також погляди замовника і розробника на нього. Однак, безпосереднього користувача ПЗ в основному цікавить експлуатаційна його якість (ISO/IEC 9126-1, 2001), тобто сукупний ефект від досягнення необхідних характеристик програми, значення яких вимірюється швидкістю та достовірністю отриманого результату, а не його властивістю. Це поняття значно ширше, ніж будь-яка окрема характеристика якості ПЗ, наприклад, зручність його використання чи надійність роботи (ISO/IEC TR 9126-4, 2004).

Рис. 1. Статичні технології оцінювання якості ПЗ

Зазвичай, під оцінюванням якості ПЗ розуміють дії, що визначають, як саме ПЗ відповідає своєму призначенню (Kuliamin, Petrenko, 2008). Якість ПЗ оцінюють з використанням моделі якості (ISO/IEC 9126-1, 2001). Таке оцінювання набуває особливого значення із розвитком і вдосконаленням технологій визначення якості ПЗ, а саме – методами метричного аналізу. Усе це призвело до потреби розроблення відповідного засобу для визначення якості ПЗ відповідними методами.

Стандарт ISO/IEC 12207 визначає не тільки основні процеси етапів розроблення ПЗ, а й організаційні та додаткові процеси, які регламентують

інженерію, планування й управління якістю ПЗ (Braude, 2004). Згідно з цим стандартом, на всіх етапах розроблення ПЗ повинен проводитися такий контроль його якості:

● перевірка відповідності користувацьких вимог до ПЗ з критеріями і показниками його якості;

● перевірка (верифікація) та атестація (валідація) проміжних результатів розроблення ПЗ на кожному з етапів реалізації програмного проекту і визначення ступеня задоволення досягнутих критеріїв і показників його якості;

● тестування готового ПЗ, збирання даних про відмови, дефекти та інші помилки, які було виявлено в ПЗ під час його безпосередньої експлуатації;

● підбір моделей надійності ПЗ за отриманими результатами його тестування (дефекти, відмови та ін.);

● перевірка досяжності критеріїв і показників якості ПЗ, заданих у вимогах до нього.

Для наочної демонстрації залежності вартості реалізації програмного проекту від рівня якості ПЗ розглянемо особливості розроблення системи захисту інформації (СЗІ), а саме її функціональну модель (рис. 2).

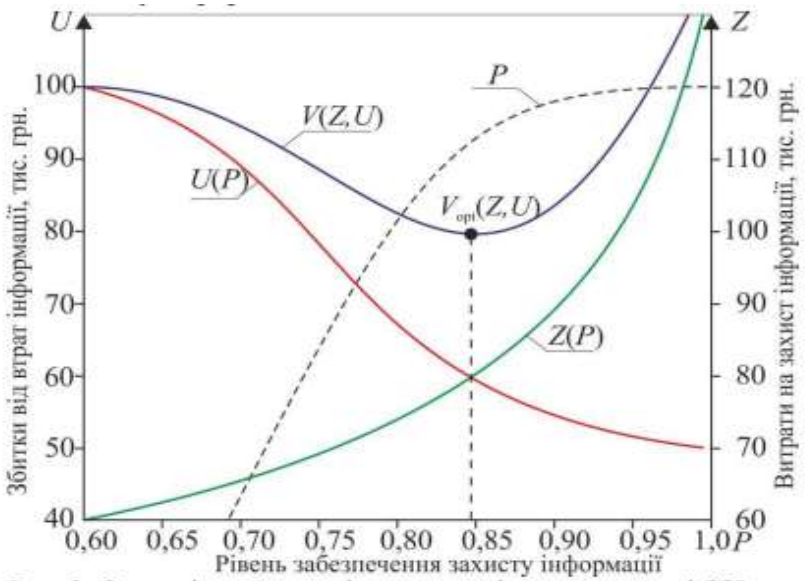


Рис. 2. Основні особливості процесу оцінювання якості СЗІ

Сутність дослідження

Цільовою програмною системою є *конструювання рівнів для 2D платформера.*

Згідно до завдання лабораторної роботи №6 було створено моделі якості заданої ПС були обрані характеристики, та відповідні їм підхарактеристики, з відповідними атрибутами для їх виміру.

Інтегральний показник якості ПЗ обчислюється за наступною формулою:

$$\begin{cases} U_{ij} = \sum_{i=1}^{I} W_i \sum_{j=1}^{J_i} W_{ij} \sum_{k=1}^{K_{ij}} W_{ijk} Q_{ijk} \\ Q = \left\{ C_i \left\{ S_{ij} \left\{ A_{ijk} \left(M_{ijk}, W_{ijk} \right) \right\}_{k=1}^{K_{ij}} \right\}_{j=1}^{J_i} \right\}_{i=1}^{I} \end{cases}.$$

Так як для вимірювання більшості метрик цільова ПС повинна бути введена в експлуатацію, на жаль на даний момент неможливо коректно обчислити показники якості ПС за заданою формулою. Але можна вказати шкалу ранжування метрик, що й буде описано в пункті 2 поточної лабораторної роботи.

Модель зовнішньої якості ПС

Розрахунок показників якості.

Побудуємо шкалу ранжування для кожної метрики:

Назва атрибуту	**Короткий опис**	**Шкала ранжування**
Відсоток реалізованих функцій	Відсоток реалізованих функцій із запланованих. Цей атрибут показує наскільки функціонально	• Плановий показник - 95-100%; • Нормативний (задовільний) показник - 80-

	повною є ПС.	95%; ● Середній показник - 50-80%; ● Незадовільний показник - 0-50%
Відношення реалізованих функцій до нереалізованих	Відношення яке показує наскільки завершеною є ПС.	● Плановий показник - 0.9-1%; ● Задовільний показник - 0.8-09%; ● Середній показник - 0.5-0.8%; ● Незадовільний показник - 0-0.5%
Стан роботи сервісу при різних стресових ситуаціях	Відсоток ситуацій в яких ПС продовжує працювати зі списку визначених критичних умов (наприклад, зникнення інтернету під час взаємодії з ПС)	● Плановий показник - 90-100%; ● Середній показник - 50-90%; ● Незадовільний показник - 0-50%
Кількість часу для відновлення роботи ПС	Середній показник часу для відновлення роботи ПС зі списку критичних умов при яких система не може працювати. Так як ступінь критичних умов може відрізнятися, потрібно створити нормативну базу знань в якій буде накопичуватися дані про вирішення проблем (в тому числі і час на її вирішення). Тому метрика немає точних рамок, а вимірюється відносно	Показник часу відносно певного типу пошкодження: ● Плановий показник - середнє значення і менше; ● Середній показник - середнє значення; ● Незадовільний показник - більше середнього значення.

	нормативної баз.	
Відгук користувачів щодо зрозумілості інтерфейсу програми	Середнє значення (від 1 - 5) оцінок користувачів, щодо розуміння та можливості орієнтуватися в інтерфейсі та функціях ПС.	• Плановий показник - 4-5; • Середній показник - 3-4; • Незадовільний показник - 1-3
Кількість годин витрачених користувачем для отримання навичок роботи з ігровою механікою	Середній показник часу в годинах, скільки часу витрачає середній користувач на отримання базових навичок з роботи ігрової механіки роботи з ПС.	Для здобуття базових навичок користування, показник має наступну шкалу: • Задовільний показник – 30 хв; • Середній показник - 1 год; • Незадовільний показник - 2 і більше годин.
Відгук користувачів щодо зовнішнього вигляду інтерфейсу програми	Середнє значення (від 1 - 5) оцінок користувачів, щодо задоволення зовнішнім виглядом інтерфейсу ПС.	• Плановий показник - 4-5; • Середній показник - 3-4; • Незадовільний показник - 1-3
Список роздільних здатностей, що підтримуються	Так як цільова ПС може запускати гру на різних моніторах, тому цей атрибут показує на скільки ПС адаптивною.	• Плановий показник - 6 найпопулярніших і більше; • Середній показник - 3 • Незадовільний показник - 1
Платформи, на яких підтримується програма	Загалом, цей атрибут показує на яких пристроях можна використовувати ПС. За ТЗ ПС повинна працювати з операційною	• Перевиконання плану - платформи які вказані в ТЗ і більше; • Плановий показник - платформи які вказані в ТЗ; • Незадовільний показник

	системою Windows.	- не підтримуються платформи які вказані в ТЗ.

Розрахунок показників якості

Експлуатаційна модель якості ПС

Згідно зі стандартом ISO/IEC 9126-4, цільова ПС має наступні характеристики експлуатаційної якості:

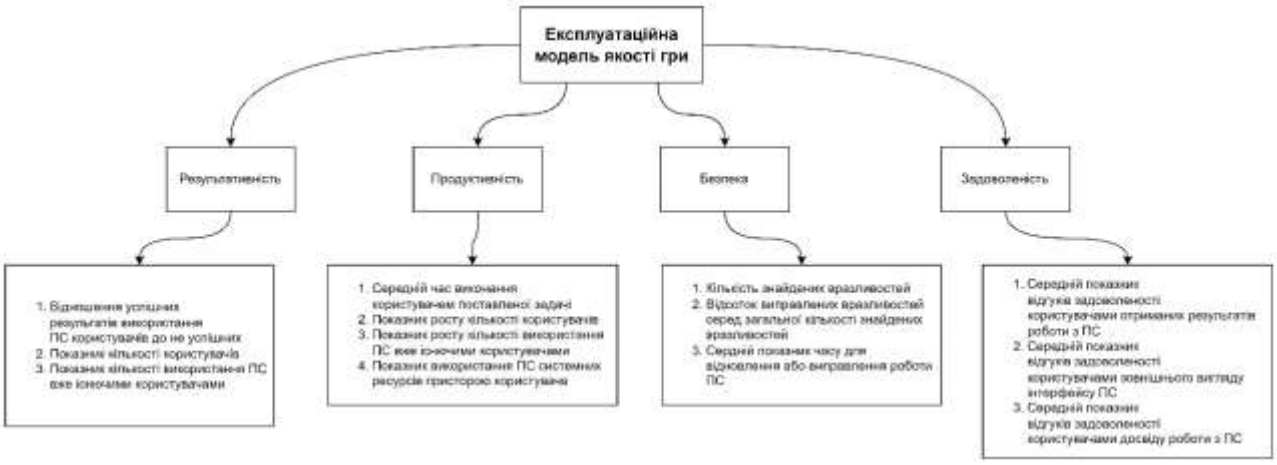
● результативність (effectiveness) – ступінь у якій користувач досягає заданих цілей по точності й повноті вирішення задач у контексті використання ПС;

● продуктивність (productivity) – ступінь у якій витрачаються ресурси на досягнення користувачем заданої ефективності у встановленому контексті використання ПС;

● безпека (safety) – рівень ризику завдання матеріальних і моральних збитків, тобто шкоди здоров'ю людей, бізнесу, майну, навколишньому середовищу при використанні ПС у встановленому контексті її використання;

● задоволеність (satisfaction) – ступінь у якій користувач задоволений ПС у певному контексті її використання.

Експлуатаційна модель якості у вигляді схеми:



Рекомендована література

1. Лавріщева К.М. Програмна інженерія. Підручн. –К.: Академперіодика, 2008. – 320 с.

2. Основы инженерии качества программных систем / Ф.И.Андон, Г.И.Коваль., Т.М.Коротун, Е.М.Лаврищева, В.Ю.Суслов. –К.: Академпериодика, 2007. –672 с.

3. Липаев В.В. Выбор и оценивание характеристик качества программных средств. Методы и стандарты. –М.: СИНТЕГ, 2001. – 228 с.

4. Основы инженерии качества программных систем / Ф.И.Андон, Г.И.Коваль., Т.М.Коротун, Е.М.Лаврищева, В.Ю.Суслов. –К.: Академпериодика, 2007. –672 с.

Лабораторна робота №8

«Дослідження принципів оцінювання внутрішньої якості програмної системи на основі стандарту ISO/IEC 9126.3 з застосуванням системи метрик М. Холстеда та Т.Маккейба»

Мета роботи:

- дослідження методів оцінки рівня якості ПС відповідно до загальної моделі якості ПС, що запропонована у стандарті ISO/IEC 9126-1;

- дослідження характеристик, підхарактеристик та метрик якості стандартів ISO/IEC 9126-1 та ISO/IEC 9126-3 для побудови внутрішньої моделі якості ПС із залученням системи метрик М.Холстеда та Т.Маккейба, особливо для прогнозування кількості дефектів та складності програмних модулів та підсистем, що функціону.nm у визначеній предметній галузі.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Ознайомитись з діючими стандартами.
4. Дослідити основні принципи методології побудови моделей якості програмних систем з використанням метрик стандарту ISO/IEC 9126.
5. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
6. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

7 Назва та мета роботи.

2. Коротко теоретичні відомості

7. Завдання дослідження лабораторної роботи згідно до обраного варіанта.

8. Результати виконаного дослідження.

9. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Основи метричної теорії програм

Метричну теорію програм представляють різні математичні моделі визначення чисельних значень характеристик програмного забезпечення, у тому числі характеристик якості.

Кожна модель представляє ту чи іншу метрику програми. По типу одержуваної інформації про метрики програм ці моделі можна розбити на наступні групи: оцінюючі відхилення від норми, а також ті, що прогнозують значення характеристик, які й формують прийняття рішення про відповідність програмного забезпечення заданим вимогам. По типу використовуваної інформації про програми в моделях розрізняють метрики, засновані на лексичному аналізі програм; на аналізі потоку управління; на аналізі модульних та міжмодульних зв'язків і метрики, а також засновані на аналізі потоку даних [1, 2, 3].

Метрики всіх цих груп використовують, головним чином, для прогнозування й оцінки складності й коректності програм. До них відносять метрики Холстеда, Джилба, Маккейба, Майерса та ін. Метрики модульних і міжмодульних зв'язків є основними характеристиками складності програмних комплексів у фазі проектування. Акумулюючи потік керування і потоки даних, ці метрики утворюють шкали функціональної міцності (зв'язності) і зчеплення модулів. Можлива непряма оцінка надійності за принципом: «чим менше складність, тим більше надійність». При цьому задача проектування надійного ПЗ зводиться до досягнення максимальної міцності і мінімального зчеплення модулів.

Історично першими з'явилися математичні моделі, що представляють метрики програм, засновані на аналізові лексики і потоку управління програм, що реалізують заданий алгоритм.

Представниками таких метрик є метрики Холстеда і Маккейба.

Метрики Холстеда

Метрики Холстеда відбивають лексичний підхід до виміру характеристик програмного забезпечення, заснований на вимірних властивостях алгоритмів. Властивості будь-якого опису алгоритму (або програми), на думку Холстеда, можуть бути виміряні чи обчислені на основі наступних метричних характеристик:

n1 **-** кількість різних (відмінних один від одного) операторів програми;

n2 **-** кількість різних (відмінних один від одного) операндів програми;

N1 **-** загальна кількість операторів програми;

N2 - загальна кількість операндів програми.

На цій основі Холстед визначає наступні метрики:

словник програми (в умовних одиницях)

$$n = n1+n2 \quad (1)$$

довжина реалізації (в умовних одиницях)

$$N = N1+N2 \quad (2)$$

довжина програми (в умовних одиницях)

$$\tilde{N} = (n1 \times \log_2 n1)+(n2 \times \log_2 n2) \quad (3)$$

обсяг програми (у бітах)

$$V = (N1+N2) \times \log_2(n1+n2) \quad (4)$$

При визначенні обсягу програми припускаємо, що позначення операндів і операторів потребує не більше одного символу.

потенційний обсяг програми

$$V^* = (n2^*+2) \times \log_2(n2^*+2), \quad (5)$$

де ***n2* -*** загальне число вхідних і вихідних параметрів.

Припускаємо, що в програмах ідеальних з погляду економії витрат пам'яті, по-перше: оператори та операнди не повторюються; по-друге: всі операнди є або вхідними, або вихідними параметрами; по-третє: для запису тексту програми досить двох операторів (опису заголовка процедури-функції і присвоювання значення). Використаємо вираз (5.4) для обсягу програми, за умови, що N1=n1=2 і N2=n2=n2* .

рівень програми (в умовних одиницях)

$$L = V^*/ V = (2 \times n2)/(n1 \times N2), \quad (6)$$

якщо n2*=2 .

рівень мови

$$\lambda = L \times V^* \quad (7)$$

інтелектуальний зміст програми (в умовних одиницях)

$$I = L \times V \cong \left(2 \times n2 / n1 \times N2\right) \times \left(N1 + N2\right) \times \log_2 \left(n1 + n2\right) \quad (8)$$

робота з програмування (в умовних одиницях)

$$E = V/L = V^2/V^* \quad (9)$$

час на програмування (в умовних одиницях)

$$T = E/S, \quad (10)$$

чи

$$T \cong \left(n1 \times N2 \times \log_2 n \times \left(n1 \times \log_2 n1 + n2 \times \log_2 n2\right)\right) / \left(n1 + n2\right), \quad (11)$$

де ***S*** – число Страуда (*5<S<20*).

Число Страуда ***S*** визначається як число «страудовських моментів» у секунду. «Страудовський момент» - це час, необхідний людині для виконання елементарного розрізнення об'єктів, подібно розрізненню кадрів

фільму. Страуд винайшов, що людина здатна розрізняти від 5 до 20 об'єктів у секунду.

Потенційний обсяг програми є мірою мінімально необхідного обсягу програми з заданим словником. Потенційний обсяг не залежить від мови реалізації. При перекладі програми з однієї мови на інший потенційний обсяг не міняється, але дійсний обсяг V чи збільшується, чи зменшується в залежності від мови реалізації.

Використовуючи вираження для потенційного обсягу програми, Холстедом отримані наступні метрики:

Рівень програми ***L<1*** характеризує ефективність реалізації алгоритму щодо витрат пам'яті. Тільки для найбільш компактної форми реалізації алгоритму ***(V=V*)*** рівень програми дорівнює 1. Всім іншим варіантам реалізації відповідають значення ***L<1***.

Рівень мови λ – це коефіцієнт пропорційності зміни обсягу програми при перекладі з однієї мови на іншу так, що добуток рівня програми на потенційний обсяг залишається незмінним.

Інтелектуальний зміст характеризує міру «сказаного» у програмі, чи її «інформативність». Інтелектуальний зміст (рівень) програми сильно корелює з потенційним обсягом ($I \approx V^*$) і теж не залежить від мови реалізації.

Робота з програмування (рівняння розумової роботи) характеризує величину розумової роботи, зв'язаної із написанням програмного коду. Оскільки сума квадратів двох величин завжди менше квадрата їхньої суми, рівняння роботи дає підставу для розбиття програми на складові частини – модулі. Модульність знижує об'єм програмування. Найбільш продуктивною є ситуація, при якій для одержання одного результату використовується не більше п'яти об'єктів. У прикладному відношенні цей результат називають гіпотезою про «шість об'єктів». Для визначення кількості модулів ***M*** у програмі Холстед рекомендує використовувати вираз:

$$M = n2^*/6, \qquad (12)$$

де n2* – загальна кількість вхідних і вихідних змінних у програмі.

З рівняння роботи отримаємо наступне рівняння помилок :

$$B = L \times E / E_0 \qquad (13)$$

де ***B*** – кількість помилок у програмі, E_0 – середнє число елементарних відмінностей між помилками програмування.

Використовуючи перетворене рівняння роботи:

$$E = \left(V^*\right)^3 / \lambda^2, \tag{14}$$

а також значення рівня англійської мови (λ=2,16), як аналог мови програмування і гіпотезу про «шість об'єктів» ідеальної по витратах пам'яті програми (n1=n1*=2, n2=n2*=6), Холстед вивів наступне рівняння для прогнозу кількості помилок у програмі:

$$B = E^{2/3}/3000, \tag{15}$$

чи:

$$B = V/3000, \tag{16}$$

де ***V*** – обсяг програми (4).

Крім свого прямого призначення ***в практичному відношенні*** метрики довжини програми (3) і довжини реалізації (2) можна використовувати для виявлення недосконалостей програмування. Якщо розрахунки довжини програми і довжини реалізації відрізняються більш ніж на ***десять відсотків***, то це свідчить про можливу наявність у програмі таких *6 класів недосконалостей:*

1. Наявність послідовності доповнюючих один одного операторів до того ж самого операнду, наприклад, A+C–A.
2. Наявність неоднозначних операндів, наприклад, A=D і A=C.
3. Наявність операндів-синонімів, наприклад, A=B и T=B.
4. Наявність загальних підвиразів: (A+B)×C + D×(A+B).
5. Непотрібне присвоювання, наприклад C=A+B, якщо змінна C використовується в програмі тільки один раз.
6. Наявність виразів, що не представлені в згорнутому виді як добуток множників, наприклад X×X+2×X×Y+Y×Y не представлено як (X+Y)×(X+Y).

Довжину реалізації ***N*** (2) можна використати для *прогнозу кількості фактичних машинних команд* ***P*** за допомогою виразу:

$$N = \frac{8}{3} \times P \tag{17}$$

чи більш грубо, за допомогою нерівності:

$$2 \times P \le T \le 4 \times P. \tag{18}$$

Рівняння роботи (5.9) можна використати для *оцінки економічної ефективності використання* тієї чи іншої мови програмування. Відносне скорочення роботи з програмування в залежності від рівня мови використовують як покажчик ефективності впровадження мови програмування у практику.

Рівень програми $0 < L \le I$ можна використовувати для оцінки *складності варіантів реалізації заданого алгоритму* ***D*** (чим менше витрат пам'яті, тим складніше варіант програми):

$$D = 1 / L \approx \frac{n1 \times N2}{2 \times n2}. \quad (19)$$

Фахівець, який добре знає мову програмування, зрозуміє програму тим швидше, чим менше її обсяг, тобто вище рівень. Але для людини, що менше разбирається в програмуванні, потрібно

більший її обсяг і менший рівень. Встановлено, що для будь-якого алгоритму, описаного різними мовами, зі збільшенням обсягу програми ***V*** рівень програми ***L*** зменшується в тій же пропорції. Тому добуток рівня програми на обсяг є постійною величиною, рівною потенційному обсягу реалізації даного алгоритму:

$$L \times V = V^* = \text{const}. \quad (20)$$

Якщо мова не міняється, а міняється тільки алгоритм, то для будь-якої мови добуток потенційного об'єму на рівень програми залишається постійною величиною, рівною рівню мови:

$$L \times V^* = \lambda = const. \quad (21)$$

Метрика Маккейба

Метрика Маккейба заснована не на аналізі лексичних особливостей програмної реалізації різноманітних алгоритмів рішення задач, а на аналізі потоку передач управління від одного оператора до іншого. Це дозволяє, на відміну від метрик Холстеда, врахувати логіку програми при оцінці її складності.

Програма (алгоритм, специфікація) має бути представлена у вигляді управляючого орієнтованого графа G=(V, E) із ***V*** вершинами і ***E*** дугами, де вершини відповідають операторам, а дуги – переходам від одного оператора до іншого. Граф, що описує програму у виді вершин-операторів і дуг-переходів, називають *графом* керування чи управляючим графом програми.

Приклад. Нехай мається програма, що зчитує із вхідного потоку символи доти, поки не буде виявлений символ «#». Текст програми мовою Сі представлений на рис.1. Відповідний граф керування програми показаний на рис.2.

Для представлення програми у виді графа необхідно визначити угоду про те, що вважати вузлом графа, бо синтаксис операторів у мовах

програмування є досить різноманітним. Звичайно враховують тільки виконувані оператори, виключаючи оператори опису даних. Бажано підібрати такі синтаксичні форми операторів, що найбільшою мірою підходять для відображення вузлом графа. Лінійні ділянки програми можна замінити одним вузлом графа. У цьому відношенні використання схем чи алгоритмів опису програм може виявитися навіть більш кращою формою опису програми, ніж текст мовою програмування. У будь-якому випадку, бажано перетворити оператори циклу в еквівалентну послідовність операторів розгалуження, додавши, при необхідності, лічильники числа повторень циклу з «верхнім» чи «нижнім» завершенням.

```
main()
{ int zeich = 'x'; while(zeich != '#')
        { printf(" Продовжити введення "); zeich =
          getchar();
        }
 printf("Введення завершене "); }
```

Рис. 1. Програма введення даних

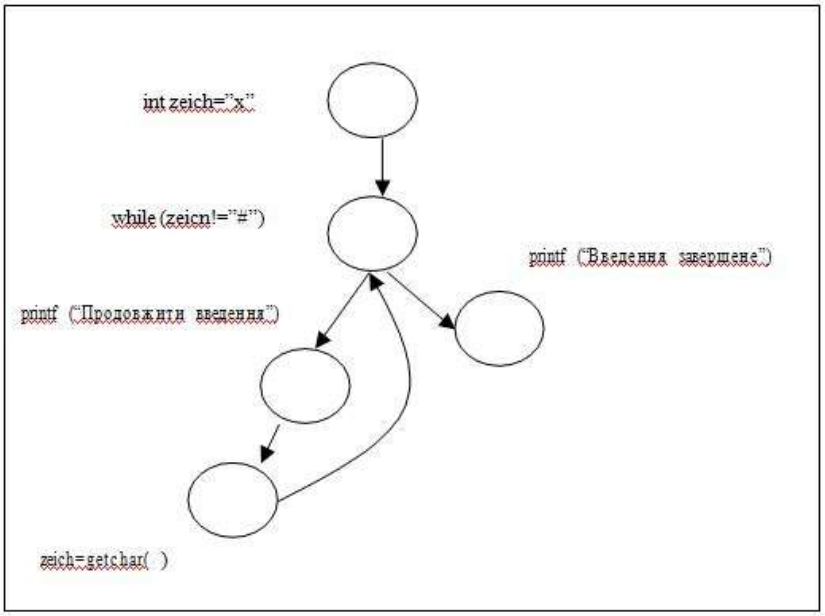


Рис. 2. Граф керування програми введення даних

Метрика Маккейба є цикломатичним числом графа передач управління програми і визначається виразом:

$$M=m-n+2, \quad (22)$$

де ***m*** – кількість ребер графа, ***n*** – кількість вершин графа. Величину *M*, розраховану по формулі (22), називають ***цикломатичним числом Маккейба***.

Теоретичною базою визначення цикломатичного числа Маккейба є теорія графів. У теорії графів цикломатичне число орієнтованого графа визначається виразом:

$$Z = m - n + 2 \times p, \quad (23)$$

де ***m*** – кількість ребер, ***n*** – кількість вершин, ***p*** – кількість компонентів зв'язності графа.

Число компонентів зв'язності ***p*** можна розглядати як мінімально необхідну кількість ребер, які потрібно додати до графа, щоб зробити його повнозвя'зним. Повнозв'язним вважають граф, у якого існує шлях з будь-якої вершини графа в будь-яку іншу вершину графа. Як показали дослідження, для графів керування програм повнозв'язність забезпечується додаванням однієї фіктивної дуги з кінцевої вершини в початкову, тобто у нашому прикладі із вершини **u** у вершину **v**.

Тому справедливо вважати, що для будь-якого графа управління програми число компонентів зв'язності дорівнює одиниці (***p=1)***. Підстановка ***p=1*** у формулу (23) дає цикломатичне число Маккейба (22). Визначимо цикломатичне число Маккейба для графа керування програми, зображеного на рис. 2. Кількість ребер графа дорівнює п'яти, кількість вершин теж дорівнює п'яти, тому цикломатичне число дорівнює: M = 5 – 5 + 2 = 2.

Фізичний зміст цикломатического числа

Для кожної дуги ***e=(v, u)| e∈E*** графа керування перша вершина (***v***) є вихідною, а друга (***u***) – кінцевою.

Шлях від однієї вершини до іншої вершини графа, якщо він складається більш ніж з однієї дуги, описують послідовністю вершин відповідних дуг ***(v, a,…,u)*** так, що вихідна вершина буде першою в перерахуванні, а кінцева – останньою.

Контур – це площина, обмежена циклічним шляхом, у якій початкова і кінцева вершина графа збігаються. Кожному контуру відповідає обмежуючий його шлях, що веде з початкової вершини графа в кінцеву.

Цикломатичне число визначає кількість незалежних контурів у повнозв'язному графі і, як наслідок, кількість різних шляхів, що ведуть з початкової вершини в кінцеву. У практичному аспекті цикломатичне число є мірою складності програми і визначає кількість тестів, достатніх для тестування за критерієм покриття всіх гілок програми. При оцінці складності програми діє наступне правило: якщо цикломатичене число більше десяти, програма має зайву складність і її варто розбити на складові частини (незалежні модулі) з меншим значенням цикломатичного числа.

Таким чином, програма організації введення даних, що має граф керування з цикломатичним числом, рівним 2, не має зайву складність і для її тестування досить підібрати два тести, що покривають гілки (a, b, c) і (a, u) (рис. 2). Помітимо, що вершини **v** і **a** графа керування програми можна об'єднати, що не приведе до зміни цикломатичного числа Маккейба.

Цикломатичне число залежить тільки від кількості управляючих операторів, що містять умовні вирази (предикати). При цьому складність предиката не враховується. Якщо в операторі циклу *while* програми організації введення даних (рис. 1) ускладнити предикат, змінивши заголовок:

while(zeich != '#') на while(zeich != '#' || zeich != '')*,

граф керування і цикломатичне число залишаться тими ж самими. Якщо програма містить тільки оператори розгалуження (без операторів вибору і переключення), цикломатичне число можна

визначити, використовуючи вираз:

$$M=u+1, \qquad (24)$$

де u – кількість операторів розгалуження.

При необхідності оператори вибору чи переключення з n гілками, можна розглядати як n операторів розгалуження.

Дослідження показують, що не існує єдиної метрики, яка б забезпечила універсальний підхід до кількісної оцінки якості ПС. Виміри й оцінка якості

дають спектр метрик, що є основою для прийняття рішень у процесі розробки і супроводження ПЗ.

Порядок виконання роботи

Вважається, що ПС створена для функціонування в межах заданої предметної області відповідно до вимог, визначених студентами у попередніх[лабораторних роботах.

В процесі виконання попередніх лабораторних робіт були побудовані дві моделі якості для оцінювання рівня якості програмної системи. Це модель зовнішньої якості ПС і модель якості у використанні (модель експлуатаційної якості ПС). При виконанні попередної лабораторної роботи був обчислений рівень якості ПП із залученням зовнішніх метрик та метрик експлуатаційної якості.

В даній лабораторній роботі необхідно визначити показники внутрішньої якості та обчислити інтегральний рівень якості ПП відповідно до стандарту ISO/IEC 9126-3 (внутрішні метрики) із залученням метрик Холстеда та Маккейба. Слід встановити взамозв'язок між внутрішніми та зовнішніми показниками властивостей ПП і доповнити модель якості ПП. Для цього виконаємо такі кроки:

1) Основну увагу необхідно зосередити на нефункціональних характеристиках ПП (надійність, зручність використання, ефективність, супроводжуваність, переносимість). Особливо важливою з них є характеристика надійність. Для усіх трьох її підхарактеристик слід розрахувати оцінки за допомогою метрик Холстеда (формули (1)-(14)) і окремо побудувати прогнозні оцінки для помилок (формули (15), (16)). Надалі шляхом обчислення мір підхарактеристик: завершеність, відмовостійкість і відновлюваність перевірити прогнозні оцінки. Міри цих підхарактеристик обчислюються відповідно до метрик, які залежать від результатів спільних перевірок програмного коду, які будемо вважати відомими (ISO/IEC 9126-3). Крім того, ці оцінки мають корелюватися із відповідними зовнішніми метриками, які можуть бути обчислені при тестуванні ПС (ISO/IEC 9126-2).

2) Із характеритики функціональність слід виконати розрахунки тільки для підхарактеристики захищеність (security). Кількість різних операторів та операндів у кожній підсистемі ПС, а також загальну їх кількість вважаємо відомою. Відомими, також, є код та документація на ПП. Складність алгоритмів оцінюється для розроблених в попередніх лабораторних роботах двох діаграм взаємодії об'єктів в межах сценаріїв, для яких виконаємо всі дії згідно підходу Маккейба для розрахунку

цикломатичного числа відповідних алгоритмів.

Приклади завдань для обчислення рівня внутрішньої якості ПС проблемної галузі:

1. Діяльність спортивного клубу

Деякий спортивний клуб має декілька команд в ігрових видах спорту, зокрема футбольну команду, яка приймає участь у чемпіонаті та проводить вдома ігри на певному стадіоні. Спортивний клуб має назву, кольори, місто базування, Раду директорів. Команда має назву, тренера, адресу базування, гравців. Кожний гравець характеризується ПІБ, амплуа (нападник, захисник і т.п.), зростом, вагою, віком. Команда проводить матчі на стадіонах, які відвідують болільники.

2. Обслуговування банківських рахунків

Клієнт має банківські рахунки в різних банках, які характеризується назвою, адресою, номером ліцензії. Кожним банком керує Рада директорів. Клієнт характеризується ПІБ,

адресою, ідентифікаційним кодом. Банківський рахунок характеризується номером, видом рахунку, сумою. Банківський рахунок можна відкрити або закрити, зняти чи переказати з нього певну суму коштів, додати готівку, нарахувати відсотки тощо. Банк також надає клієнтам кредити, які характеризуються назвою, типом, датами видачі та погашення, річним відсотком.

3. Діяльність проектної установи

Проектна установа ліцензована на розробку архітектурних проектів і складається з декількох відділів. Кожен відділ очолює завідувач і в ньому працюють співробітники на певних посадах. Відділи мають назву та розташовані в певних приміщеннях, де маються телефони. Співробітники виконують проекти, котрі мають назву, дату початку і закінчення, а також керівника. Один співробітник може виконувати декілька проектів. Виконані проекти розглядаються та затверджуються Державною архітектурною комісією, котра розглядає проекти в зазначені терміни.

4. Діяльність загальноосвітньої школи

Загальноосвітні школи, в яких навчаються учні, характеризуються номером, назвою, адресою, ПІБ директора. В школах мається певна кількість класів, котрі мають назву, класного керівника, список учнів. Дисципліни викладаються педагогами, причому один вчитель може викладати декілька предметів, а однакові дисципліни можуть викладатися різними вчителями. Предмети викладаються згідно розкладу у кабінетах,

котрі мають номер, назву, відповідне обладнання. Предмети мають назву, кількість годин вивчення, список навчальних посібників.

Рекомендована література

1. Брауде Э. Технология разработки программного обеспечения. – СПб.: Питер, 2004. – 655 с.

2. Лавріщева К.М. Програмна інженерія. Підручн. –К.: Академперіодика, 2008. – 320 с.

3. Соммервилл И. Инженерия программного обеспечения. – М.: Изд. дом Вильямс, 2002. – 624 с.

Приклад виконання лабораторної роботи

Цільовою програмною системою є **веб-редактор тривимірних сцен.**

1. Розрахуємо метрики Холстеда

Розрахунки проведемо засобами мови програмування python.

Вхідні дані:

In [1]:
```
import math
from IPython.display import Math

n1 = 28 # кількість різних (відмінних один від одного) операторів програми
n2 = 547 # кількість різних (відмінних один від одного) операндів програми
N1 = 13474 # загальна кількість операторів програми
N2 = 984 # загальна кількість операндів програми
```

Словник програми:

In [2]:
```
n = n1 + n2
Math(fr"\large n = {round(n, 0)}")
```

Out[2]: $n = 575$

Довжина реалізації програми:

In [3]:
```
N = N1 + N2
Math(fr"\large N = {round(N, 0)}")
```

Out[3]: $N = 14458$

Довжина програми:

In [4]:
```
N_ = (n1 * math.log2(n1)) + (n2 * math.log2(n2))
Math(r"\large \overline{N} = " + fr"{round(N_)}")
```

Out[4]: $\overline{N} = 5110$

Об'єм:

In [5]:
```
V = N * math.log2(n)
Math(fr"\large V = {round(V / 8)}byte")
```

Out[5]: $V = 16568byte$

Потенційний обсяг програми є мірою мінімально необхідного обсягу програми з заданим словником. Потенційний обсяг не залежить від мови реалізації. При перекладі програми з однієї мови на інший потенційний обсяг не міняється, але дійсний обсяг **V** чи збільшується, чи зменшується в залежності від мови реалізації.

Потенційний об'єм:

In [6]:
```
V_pt = (n2 + 2) * math.log2(n2 + 2)
Math(fr"\large V^* = {round(V_pt)}bit")
```

Out[6]: $V^* = 4996bit$

Використовуючи вираження для потенційного обсягу програми, Холстедом отримані наступні метрики:

Рівень програми $L<1$ характеризує ефективність реалізації алгоритму щодо витрат пам'яті. Тільки для найбільш компактної форми реалізації алгоритму ($V=V^*$) рівень програми дорівнює 1. Всім іншим варіантам реалізації відповідають значення $L<1$.

Рівень програми:

In [11]:
```
V_d = (N1 + N2) * math.log2(n1 + n2)
V_d_st = (n2 + 2) * math.log2(n2 + 2)
L = V_d_st / V_d
Math(fr"\large L = {round(L, 2)}")
```

Out[11]: $L = 0.04$

Рівень мови – це коефіцієнт пропорційності зміни обсягу програми при перекладі з однієї мови на іншу так, що добуток рівня програми на потенційний обсяг залишається незмінним.

Рівень мови:

In [12]:
```
lang_level = L * V_pt
Math(fr"\large \lambda = {round(lang_level)}")
```

Out[12]: $\lambda = 188$

Інтелектуальний зміст характеризує міру «сказаного» у програмі, чи її «інформативність». Інтелектуальний зміст (рівень) програми сильно корелює з потенційним обсягом (I ≈V*) і теж не залежить від мови реалізації.

Інтелектуальний зміст:

In [13]:
```
I = L * V
Math(fr"\large I = {round(I)}")
```

Out[13]: $I = 4996$

Робота з програмування (рівняння розумової роботи) характеризує величину розумової роботи, пов'язаної із написанням програмного коду. Оскільки сума квадратів двох величин завжди менше квадрата їхньої суми, рівняння роботи дає підставу для розбиття програми на складові частини – **модулі**.

Модульність знижує об'єм програмування. Найбільш продуктивною є ситуація, при якій для одержання одного результату використовується не більше п'яти об'єктів. У прикладному відношенні цей результат називають гіпотезою про «шість об'єктів».

Визначення кількості модулів:

In [14]:
```
M = n2/6
Math(fr"\large M = {round(M)}")
```

Out[14]: $M = 91$

Обчислення складності, зусилля, та час для програмування:

Складність:

In [7]:
```
D = n1 / 2 + N2 / n2
Math(fr"\large D = {round(D)}")
```

Out[7]: $D = 16$

Зусилля:

In [8]:
```
E = D * V
Math(fr"\large E = {round(E)}")
```

Out[8]: $E = 2094027$

Час, необхідний для програмування:

In [9]:
```
T = E / 18
Math(fr"\large T = {round(T)} = {round(T / 60 / 60)} \space год")
```

Out[9]: $T = 116335 = 32 \text{ год}$

Оцінка кількості помилок:

In [10]:
```
B = V / 3000
Math(fr"\large B = {round(B)}")
```

Out[10]: $B = 44$

2. Цикломатичне число Маккейба

В минулих роботах було визначено алгоритм взаємодії об’єктів в межах сценаріїв. А саме:

При оцінці складності програми діє наступне правило: якщо цикломатичне число більше десяти, програма має зайву складність і її варто розбити на складові частини (незалежні модулі) з меншим значенням цикломатичного числа.

Розрахуємо цикломатичне число Маккейба:

Позначимо вихідні дані:

In [15]:
```
m = 5 # кількість ребер графа
n = 4 # кількість вершин графа
```

Цикломатичним числом Маккейба:

In [16]:
```
M = m - n + 2
Math(fr"\large M = {M}")
```

Out[16]: $M = 3$

Лабораторна робота №9

«Дослідження методів забезпечення якості при колективному розробленні програмних систем»

Мета роботи: засвоєння базових знань щодо основних положень командного розроблення програмного забезпечення (ПЗ), управління життєвим циклом (ЖЦ) додатків, гнучкої методології створення програмних систем, а також можливості інструментарію VisualStudio і TeamFoundationServer для управління ЖЦ додатків.

Підготовка до роботи: Вивчити і уяснити призначення і зміст,завдання, інструментальні засоби та основні процеси при колективній розробці складних програмних засобів (ПЗ).

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Ознайомитись з діючими стандартами.
4. Для проведення досліджень встановити і налагодити технологічне програмне забезпечення в обчислювальному середовищі віртуальних машин.
5. Дослідити методології розроблення ПЗ Microsoft Solutions Framework (MSF), принципів створення бібліотеки MSF, моделі команди в MSF, рольових кластерів, масштабованості команд та керування компромісами у MSF.
6. Дослідити основні принципи гнучкого підходу та гнучкого розроблення до створення ПЗ.
7. Дослідити основні принципи реалізації концепції керування програмним проектом на всіх етапах життєвого циклу у Visual Studio.
8. Дослідити функціональні можливості та архітектури TeamFoundationServe (TFS) (посилання на Azure DevOps Server https://learn.microsoft.com/ru-ru/azure/devops/server/download/azuredevopsserver?source=recommendations&view=azure-devops
9. Дослідити способи розгортання TFS на одному або декількох серверах, в одному домені, робочі групі або в декількох доменах, шаблонів командних проектів TFS, області керування командними проектами.

10. Дослідити питання створення командного проекту, зміст програмної інфраструктури проекту, склад і призначення робочих елементів.

11. Провести аналіз методології Scrum, робочі елементи шаблону Microsoft Visual Studio Scrum.

12. За результатами досліджень скласти звіт з обґрунтованими висновками.

13. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..

14. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

8 Назва та мета роботи.

2. Коротко теоретичні відомості

15. Завдання дослідження лабораторної роботи згідно до обраного варіанта.

16. Результати виконаного дослідження.

17. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Раціональна організація процесів розроблення ПЗ описується в стандартах (міжнародних, державних, копроративних), які часто називають методологіями розроблення ПЗ. Методології створення ПЗ, як правило, розробляються провідними виробниками програмних систем та їх спільнотами з врахуванням особливостей програмних продуктів, а також сфери впровадження. Методології описують підходи до організації раціональної стратегії та можливого набору процесів створення ПЗ.

Наразі все більше поширення одержують гнучкі методології розроблення ПЗ, де основна увага зосереджена на створенні якісного продукту, а не на підготовці вичерпної документації за проектом. При цьому акцент робиться на організації ефективного управління командою. Як зазначає Еріх Гамма, «ключ до своєчасної поставки продукту - не процеси, а люди». Самоорганізація та цілеспрямованість команди розробників дозволяє створювати високоякісні програмні продукти в стислі терміни.

Інструменти управління ЖЦ додатків в основному сприяють успішності програмних проектів. Компанія Microsoft надає розробникам гнучкий інструментарій для управління ЖЦ додатків - ALMVisualStudio і TeamFoundationServer. Традиційні засоби розроблення програм у VisualStudio доповнені засобами архітектурного проектування і тестування. Інструментарій TeamFoundationServer дозволяє формувати та відстежувати вимоги до програмної системи, зв'язувати їх із задачами та реалізацією, розподіляти між членами команди, проводити побудову програмного продукту, керувати тестуванням, проводити контроль версій надавати засоби комунікації з членами команди та замовниками, підготовляти численні звіти.

1. Методологія розроблення ПЗ Microsoft Solutions Framework (MSF). Принципи створення бібліотеки MSF. Модель команди в MSF, рольові кластери, масштабованість команд та керування компромісами у MSF

Microsoft Solutions Framework (MSF) є методологією розроблення ПЗ, яка представляє собою узагальнення кращих проектних практик, які використовувались командами розробників Microsoft. Дана методологія описує керування людьми та робочими процесами при розробленні IT-рішень.

IT-рішення - розуміється як скоординоване постачання набору елементів (таких як програмні засоби, документація, навчання та супровід), необхідних для задоволення бізнес-потреб конкретного замовника.

Концепція управління ЖЦ додатків, прийнята розробниками ПЗ, призвела до того, що методологія MSF стала складовою частиною продукту Visual Studio Team System (VSTS), який реалізовував підхід Microsoft. До продукту VSTS ввійшли шаблони процесів для реалізації положень моделі CMMI - MSF for CMMI і моделей гнучкого розроблення ПЗ - MSF for Agile та Scrum. Таким чином, VSTS є інструментарієм управління ЖЦ додатків, який дозволяє створювати програмні системи різного призначення в командах, які дотримуються різних підходів до управління процесами розроблення ПЗ.

Основними є наступні принципи MSF:

єдине бачення проекту, яке передбачає розуміння всіма зацікавленими особами цілей та задач створення ПЗ;

гнучкість - готовність до змін, що забезпечує можливість уточнення та зміни вимог в процесі розроблення ПЗ, оперативного та швидкого реагування на поточні зміни умов проекту при незмінній ефективності управлінської діяльності;

концентрація на бізнес-пріоритетах, що передбачає створення продукту з високою якістю для споживача та формування певної вигоди або віддачі. Для організацій, як правило, це одержання прибутку;

заохочення вільного спілкування, що передбачає відкритий та чесний обмін інформацією як всередині команди, так і з ключовими зацікавленими особами.

Універсальність моделі MSF (рис. 1) визначається тим, що завдяки своїй гнучкості та відсутності жорстко встановлених зв'язків і процедур, вона може бути застосована при розробленні різних програмних додатків, які можуть використовуватись в бізнесі та повсякденному житті.

Розроблення

В основі методології MSF лежить ітеративний інтегрований підхід до створення та впровадження рішень, який базується на фазах та віхах.

Ітеративність підходу передбачає поетапне створення робото здатної програмної системи з визначеною функційністю,

Модель MSF базується на сполученні двох моделей ЖЦ програмних систем: каскадної та спіральної (рис. 1). яка відбиває вимоги до кінцевого продукту на даному етапі розроблення. Кожен виток спіралі складається з ідентичних фаз, на яких виконуються етапи робіт з формування концепції фази, планування, розроблення, стабілізації та впровадження. Набір вимог програмної системи та відповідних їм задач, які реалізуються на заданому витку спіралі, визначається менеджером проекту на основі ранжування вимог

до системи. Кожен наступний виток спіралі додає до програмної системи функційність, яка відбиває вимоги замовника.

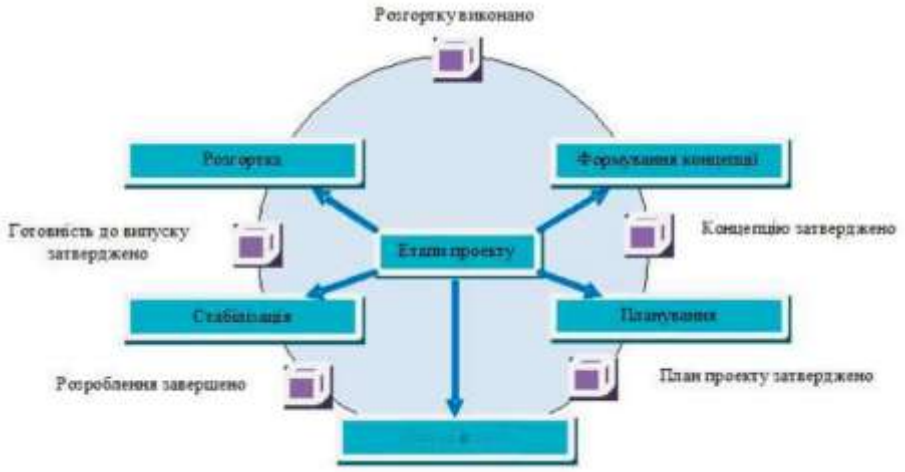


Рис.1 - Модель життєвого циклу рішення MSF

Інтеграція в межах одного проекту процедур розроблення та впровадження системи дозволяє представити замовнику різні проміжні робото здатні версії програмного продукту, оперативно вносити зміни, які відбивають бачення замовником функційності та дизайну проектованої системи, знизити ризики проекту за рахунок раннього виявлення проблем та можливості своєчасного їх усунення.

Фази проекту визначають послідовно вирішувані задачі, а віхи (milestones) - ключові точки проекту, які характеризують досягнення будь-якого істотного результату.

У MSF використовуються два види віх: головні та проміжні. Вони мають наступні характеристики:

головні віхи служать точками переходу від однієї фази до іншої і визначають зміни в поточних задачах ролевих кластерів проектної команди; в MSF головні віхи є достатньо універсальними для застосування в будь-якому IT-проекті;

проміжні віхи показують досягнення визначеного прогресу у виконанні фази проекту і розчленовують великі сегменти роботи на менші ділянки, які піддаються огляду та керуванню; проміжні віхи можуть варіюватись в залежності від характеру проекта.

Головна особливість моделі команди у MSF є те, що вона не має офіційного лідера. Всі відповідають за проект в рівному ступені, рівень зацікавленості кожного в результаті дуже високий, а комунікації всередині групи чіткі, ясні, дружні та відповідальні. Таке можливе при високому рівні

самосвідомості та зацікавленості кожного члена команди, а також при достатньо високому рівні професіоналізму.

Однією з особливостей відношень всередині команди є висока культура дисципліни обов'язків:

готовність працівників приймати на себе обов'язки перед іншими;

чітке визначення тих обов'язків, які вони на себе беруть;

прагнення прикладати необхідні зусилля до виконання своїх обов'язків;

готовність чесно та невідкладно інформувати про загрози виконання своїх обов'язків.

Рольові кластери MSF засновані на семи якісних цілях, досягнення яких визначає успішність проекту. Ці цілі обумовлюють модель проектної групи та утворюють рольові кластери (або просто ролі) в проекті. Кожен кластер може включати одного або декількох фахівців. Кожний рольовий кластер представляє унікальну точку зору на проект, і в той же час жоден з членів проектної групи не в стані сам успішно представляти всі можливі погляди, які відбивають якісно різні цілі.

У MSF є наступні рольові кластери:

1) управління продуктом - основна задача кластеру забезпечити, щоб замовник залишився задоволеним в результаті виконання проекту; цей рольовий кластер в проекті представляє інтереси замовника та бізнес-сторону проекту та забезпечує його узгодженість зі стратегічними цілями замовника;

2) управління програмою - кластер забезпечує управлінські функції - відстежування планів та їх виконання, відповідальність за бюджет, ресурси проекту, вирішення проблем та труднощів процесу, створення умов, за яких команда може працювати ефективно, долаючи мінімум імамкратичних перешкод;

3) розроблення - кластер забезпечує розроблення коду додатку;

4) тестування - кластер відповідає за тестування ПЗ;

5) задоволення споживача - кластер розв'язує задачі дизайну інтерфейсу користувача та забезпечення зручності експлуатації ПЗ, навчання всіх користувачів роботі з ПЗ, створення документації для користувача;

6) управління випуском - кластер відповідає за впровадження проекту та його функціювання, бере на себе зв'язок між розробленням рішення, його впровадженням та наступним супроводом, забезпечуючи інформованість членів проектної групи про наслідки їхніх рішень;

7) архітектура - кластер відповідає за організацію та виконання високорівневого проектування рішення, створення функційної

специфікації ПЗ та керування цією специфікацією в процесі розроблення, визначення меж проекту та ключових компромісних рішень.

Зміни в задачах рольових кластерів проектної команди відбуваються із змінами фаз проекту. Перехід від однієї фази до іншої включає в себе також перенос основної відповідальності від одних рольових кластерів до інших.

В залежності від розміру та складності проекту модель команд MSF припускає масштабування. Для невеликих та нескладних проектів один співробітник може об'єднувати декілька ролей. При цьому деякі ролі не можна об'єднувати. Зокрема, не можна суміщати розроблення та тестування, оскільки необхідно, щоб у тестувальників було сформовано свій, незалежний, погляд на систему, який базується на вивченні вимог. В таблиці 1 представлені рекомендації MSF відносно суміщення ролей в межах одного проекту одним членом команди. «+» означає, що суміщення можливе, «+-» - що суміщення можливе, але небажане, «-» означає, що суміщення не рекомендується.

Для великих команд (більше 10 чоловік) модель проектної групи MSF пропонує розбиття на малі багатопрофільні групи напрямків. Ці малі колективи працюють паралельно, регулярно синхронізуючи свої зусилля, кожна з яких базується на основі моделі кластерів. Такі команди мають чітко визначену задачу і відповідальні за всі питання, які до неї належать, починаючи від проектування та складання календарного графіку.

Таблиця 1 - Рекомендації MSF відносно суміщення ролей в команді проекту

	Управління продуктом	Управління програмою	Розроблення	Тестування	Задоволення користувача	Управління випуском	Архітектура
Управління продуктом		-	-	+	+	+ -	-
Управління програмою	-		-	+-	+-	+	+
Розроблення	-	-		-	-	-	+
Тестування	+	+-	-		+	+	+-
Задоволення користувача	+	+-	-	+		+ -	+-
Управління випуском	+-	+	-	+	+-		+

Архітектура	-	+	+	+-	+-	+	

Крім того, коли рольовому кластеру необхідно багато ресурсів, формуються так звані функційні групи, які потім об'єднуються в рольові кластери. Вони створюються у великих проектах, коли необхідно згрупувати працівників всередині рольових кластерів за їх галузями компетенції. Часто функційні групи мають внутрішню ієрархічну структуру. Наприклад, менеджери програми можуть бути підзвітними провідним менеджерам програми, які, в свою чергу, звітують перед головним менеджером програми. Подібні структури можуть також з'являтись всередині галузей компетенцій.

Керування компромісами. Добре відома взаємозалежність між ресурсами проекту (людськими та фінансовими), його календарним графіком (часом) та реалізованими можливостями (функційність). Ці три змінні утворюють трикутник, показаний на рис. 2. Після досягнення рівноваги в цьому трикутнику зміна будь-якої з його сторін для підтримки балансу вимагає модифікацій на іншій (двох інших) сторонах та/або на зміненій стороні.

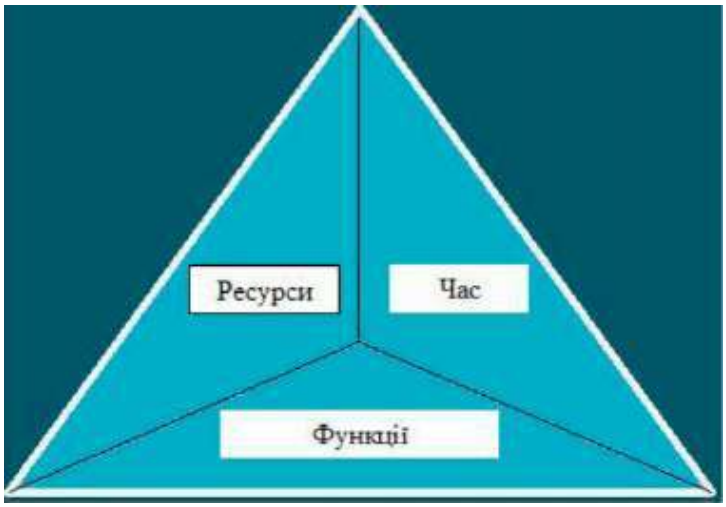


Рис. 2 - Трикутник компромісів

В проектній практиці може фіксуватись один з ресурсів, тоді в процесі проектування системи вимірюванню підлягають лише два ресурси, що залишились. Дана ситуація задається матрицею компромісів, приведеною на рис. 3. Так, якщо фіксуються ресурси, то час виконання проекту узгоджується, а функційні можливості підлягають зміні.

УзгоджусяФіксується	Приймається
*	
*	
*	

Рис. 3 - Матриця компромісів

2. Гнучкий підхід до створення ПЗ, основні принципи гнучкого розроблення

Гнучка методологія розроблення ПЗ орієнтована на використання ітеративного підходу, при якому програмний продукт створюється поступово, невеликими кроками, які включають реалізацію певного набору вимог. При цьому передбачається, що вимоги можуть змінюватись. Команди, які використовують гнучкі методології, формуються з універсальних розробників, які виконують різні задачі в процесі створення програмного продукту.

При використанні гнучких методологій мінімізація ризиків здійснюється шляхом зведення розроблення до серії коротких циклів, які називаються ітераціями, тривалістю 2-3 тижні. Ітерація представляє собою набір задач, запланованих на виконання в певний період часу. В кожній ітерації створюється роботоздатний варіант програмної системи, в якій реалізуються найбільш пріоритетні (для даної ітерації) вимоги замовника. На кожній ітерації виконуються всі задачі, необхідні для створення роботоздатного ПЗ: планування, аналіз вимог, проектування, кодування, тестування та документування. Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі, що поточний програмний продукт готовий до випуску в кінці кожної ітерації. По завершенню кожної ітерації команда виконує переоцінку пріоритетів вимог до програмного продукту, можливо, вносить корективи у розроблення системи.

Для методології гнучкого розроблення декларовані ключові постулати, які дозволяють командам досягати високої продуктивності:

- люди та їх взаємодія;
- оставка працюючого ПЗ;
- співробітництво із замовником;
- реакція на зміни.

Люди та взаємодія. Люди - найважливіша складова частина успіху. Окремі члени команди та якісні комунікації важливі для високопродуктивних команд. Для сприяння комунікації гнучкі методи передбачають часті обговорення результатів роботи та внесення змін в рішення. Обговорення можуть проводитись щоденно тривалістю по декілька хвилин і по завершенню кожної ітерації - аналізом результатів робіт та ретроспективою. Для ефективних комунікацій при проведенні зборів учасники команд повинні дотримуватись наступних ключових правил поведінки:

- думки кожного учасника команди;

- бути правдивим при будь-якому спілкуванні;
- прозорість всіх даних, дій та рішень;
- впевненість, що кожен учасник підтримає команду;
- відданість команді та її цілям.

Для створення високопродуктивних команд в гнучких методологіях, крім ефективної команди та якісних комунікацій, необхідний досконалий програмний інструментарій.

Працююче ПЗ важливіше за всеосяжну документацію. Всі гнучкі методології виділяють необхідність доставки замовнику невеликих фрагментів працюючого ПЗ через задані інтервали. ПЗ, як правило, повинне пройти рівень модульного тестування, тестування на рівні системи. При цьому обсяг документації повинен бути мінімальним. В процесі проектування команда повинні підтримувати в актуальному стані короткий документ, який містить обґрунтування розв'язку та опис структури.

Співробітництво із замовником важливіше формальних домовленостей за контрактом. Щоб проект успішно завершився, необхідне регулярне та часте спілкування із замовником. Замовник повинен регулярно приймати участь в обговоренні прийнятих рішень з ПЗ, висловлювати свої побажання та зауваження. Залучення замовника у процес розроблення ПЗ необхідне для створення якісного продукту.

Оперативне реагування на зміни важливіше за слідування планом. Здатність реагування на зміни багато в чому визначає успішність програмного проекту. В процесі створення програмного продукту дуже часто змінюються вимоги замовника. Замовники дуже часто точно не знають, чого хочуть, доки не побачать працююче ПЗ. Гнучкі методології шукають зворотній зв'язок віз замовників в процесі створення програмного продукту. Оперативне реагування на зміни необхідне для створення продукту, який задовольнить замовника та забезпечить цінність для бізнесу.

Постулати гнучкого розроблення підтримуються 12 принципами. В конкретних методологіях гнучкого розроблення визначено процеси та правила, які в більшому або меншому степені відповідають цим принципам. Гнучкі методології створення ПЗ базуються на наступних принципах:

1) вищий пріоритет надавати задоволенню побажань замовника за допомогою поставки корисного ПЗ в стислі терміни з наступним неперервним оновленням. Гнучкі методики передбачають швидку поставку початкової версії та часті оновлення. Метою команди є поставка робото здатної версії протягом декількох тижнів з моменту початку проекту. Далі програмні системи з поступово нарощуваною функційністю повинні постачатись кожні декілька тижнів. Замовник може почати промислову

експлуатацію системи, якщо вважатиме, що вона достатньо функційна. Також замовник може просто ознайомитись з поточною версією ПЗ, надати свій відгук із зауваженнями;

2) ігнорувати зміну вимог, нехай навіть на пізніх етапах розроблення. Гнучкі процеси дозволяють враховувати зміни для забезпечення конкурентних переваг замовника. Команди, які використовують гнучкі методики, прагнуть зробити структуру програми якісною, з мінімальним впливом змін на систему в цілому;

3) постачати нові працюючі версії ПЗ часто, з інтервалом від одного тижня до двох місяців, надаючи перевагу меншим термінам. При цьому ставиться мета постачати програму, яка задовольняє потреби користувача, з мінімумом супровідної документації;

4) замовники і розробники повинні працювати сумісно протягом всього проекту. Вважається, що для успішного проекту замовники, розробники та всі зацікавлені особи повинні спілкуватись часто і багато для цілеспрямованого вдосконалення програмного продукту;

5) проекти повинні втілювати в життя цілеспрямовані люди. Створіть команді проекту нормальні умови роботи, забезпечте необхідну підтримку та сподівайтесь, що члени команди доведуть справу до завершення;

6) найефективніший та найпродуктивніший метод передачі інформації команді розробників та обміну думками всередині неї - розмова обличчям до обличчя. В гнучких проектах основний спосіб комунікації - просте людське спілкування. Письмові документи створюються та оновлюються поступово по мірі розроблення ПЗ і лише за необхідності;

7) працююча програма - основний показник прогресу в проекті. Про наближення гнучкого проекту до завершення роблять висновки за тим, наскільки наявна в даний момент програма відповідає вимогам замовника;

8) гнучкі процеси сприяють довготерміновому розробленню. Замовники, розробники та користувачі повинні бути в стані підтримувати незмінний темп як завгодно довго;

9) незмінна увага до технічної досконалості та якісного проектування підвищує віддачу від гнучких технологій. Члени гнучкої команди прагнуть створювати якісний код, регулярно проводячи рефакторинг;

10) простота - мистецтво досягати більшого, роблячи менше. Члени команди вирішують поточні задачі максимально просто та якісно. Якщо в майбутньому виникне якась проблема, то в якісний код є можливість внести зміни без великих витрат;

11) найкращі архітектури, вимоги та проекти видають команди, які самостійно організуються. В гнучких командах задачі доручаються не

окремим членам, а команді в цілому. Команада самостійно вирішує, як найкраще реалізувати вимоги замовника. Члени команди сумісно працюють над всіма аспектами проекту. Кожному учаснику дозволено вносити свій внесок у спільну справу. Немає такого члену команди, який одноосібно відповідав би за архітектуру, вимоги або тести;

12) команда повинна регулярно замислюватись над тим, як стати ще більш ефективною, а потім відповідно коригувати та підлаштовувати свою поведінку. Гнучка команда постійно коригує свою організацію, правила, угоди та стосунки.

Вищенаведеним принципам відповідають низка методологій розроблення ПЗ:

- AgileModeling - набір понять, принципів та прийомів (практик), які дозволяють швидко та просто виконувати моделювання та документування в проектах розроблення ПЗ;
- AgileUnifiedProcess(AUP) - спрощена версія IBM RationalUnifiedProcess(RUP), яка описує просте та зрозуміле наближення (модель) для створення ПЗ бізнес-додатків;
- OpenUP - це ітеративно-інкрементальний метод розроблення ПЗ. Позиціонується як легкий та гнучкий варіант RUP;
- AgileDataMethod - група ітеративних методів розроблення ПЗ, в яких вимоги та рішення досягаються в рамках співробітництва різних крос-функційних команд;
- DSDM - методика розроблення динамічних систем, заснована на концепції швидкого розроблення додатків (RapidApplicationDevelopment, RAD). Представляє собою ітеративний та інкрементний підхід, який надає особливого значення тривалій участі в процесі користувача/споживача;
- Extremeprogramming (XP) - екстремальне програмування;
- Adaptive software development (ADD) - адаптивне розроблення програм;
- Featuredrivendevelopment (FDD) - розроблення, орієнтоване на поступове нарощування функційності;
- GettingReal - ітеративний підхід без функційних специфікацій, який використовується для web-додатків;
- MSFfogAgileSoftwareDevelopment - гнучка методологія розроблення ПЗ компанії Microsoft;
- Scrum - встановлює правила керування процесом розроблення та дозволяє використовувати вже існуючі практики кодування, з коригуванням вимог або з внесенням тактичних змін.

Слід зазначити, що в чистому вигляді методології гнучкого програмування рідко використовуються командами розробників. Як правило, успішні команди застосовують корисні прийоми та властивості декількох процесів, підлаштовуючи їх під конкретне уявлення команди про гнучкість процесу розроблення.

3. Реалізація концепції керування програмним проектом на всіх етапах життєвого циклу у Visual Studio

Управління життєвим циклом додатків (application lifecycle management - ALM) - це концепція управління програмним проектом на всіх етапах його життя. Для реалізації цієї концепції компанія Microsoft пропонує рішення на основі VisualStudio та TeamFoundationServer (TFS). Технології ALM у Visual Studio дозволяють розробникам контролювати життєвий цикл створення ПЗ, скорочуючи час розроблення, усуваючи видатки та впроваджуючи неперервний цикл реалізації бізнес- цінностей.

Управління життєвим циклом додатку у Visual Studio базується на наступних принципах:

- продуктивність (productivity);
- інтеграція (integration);
- можливість розширення (extensibility).

Продуктивність забезпечується можливістю сумісної

роботи та керуванням складністю продукту. Всі елементи проекту (вимоги, задачі, тестові випадки, помилки, код і побудови) та звіти централізовано керуються через TFS. Інструменти візуального моделювання архітектури, можливості управління якістю коду, інструменти тестування дозволяють керувати складністю продукту.

Інтеграція забезпечується можливостями Visual Studio щодо надання всім учасникам проекту інформації про стан справ, що спрощує комунікацію між членами команди та забезпечує прозорість ходу процесу проектування.

Можливість розширення забезпечується API-інтерфейсом служб TFS та інтегрованим середовищем розроблення (integrated development environment - IDE). API-інтерфейс служб TFS дозволяє створювати власні інструменти та розширяти існуючі, а IDE - кінцевим користувачам та стороннім розробникам додавати інструменти з додатковими функціями.

При створенні програмного продукту необхідно спочатку спроектувати архітектуру, що покладається на архітектора програмного продукту. На основі архітектури здійснюється розроблення, що є призначенням розробника ПЗ. Створений продукт необхідно тестувати на його відповідність вимогам замовника, що здійснює тестувальник. Visual Studio і

TFS забезпечують сумісну командну роботу архітектора, розробника та тестувальника, надаючи їм необхідний інструментарій та функційні можливості для виконання необхідних робіт.

У Visual Studio для архітектурного проектування використовуються інструменти візуального проектування на основі мови UML, призначені для:

- візуалізації архітектурних аспектів проектованої системи;
- створення моделей структури та поведінки системи;
- розроблення шаблонів для проектування системи;
- документування прийнятих рішень.

Діаграми UML дозволяють візуально описувати додаток, наочно представляти архітектуру та документувати вимоги до додатка.

Архітектурні інструменти у Visual Studio 2012 Ultimate дозволяють створювати шість видів схем та документ- орієнтованих графів:

схема класів UML;

- схема послідовностей UML;
- схема варіантів використання UML;
- схема активності UML;
- схема компонентів UML;
- схема шарів.

Схеми (діаграми) класів UML описують об'єкти в прикладній системі. Діаграми класів відбивають ієрархію всередині додатку або системи та зв 'язки між ними.

Схеми (діаграми) послідовностей UML показують взаємодію між різними об'єктами. Вони використовуються для демонстрації взаємодії між класами, компонентами, підсистемами або суб'єктами.

Схеми (діаграми) варіантів використання UML визначають функційність системи та описують з точки зору користувачів їх можливі дії з програмним продуктом. Дані діаграми визначають зв'язки між функційними вимогами, користувачами та основними компонентами системи.

Схеми (діаграми) активності UML описують бізнес-процес або програмний процес у вигляді потоку робіт через послідовні дії. Діаграми активності використовуються для моделювання логіки в конкретному варіанті використання або для моделювання подробиць бізнес-логіки.

Схеми (діаграми) компонентів UML описують розподіл програмних складових додатку, дозволяючи наочно відобразити на високому рівні структуру компонентів та служб. За допомогою цих схем можна візуалізувати компоненти та інші системи, показуючи зв'язки між ними. В якості компонентів можуть виступати виконавчі модулі, DLL-бібліотеки та інші системи.

Схеми (діаграми) шарів використовуються для опису логічної архітектури системи. Діаграми шарів можуть використовуватись для перевірки того, що розроблений код відповідає високорівневому проекту на схемі шарів. Діаграми дозволяють перевіряти архітектуру додатка на відповідність базі коду.

Документ-орієнтовані графи дозволяють створити граф залежностей, який відбиває відношення між компонентами архітектурних артефактів. Графи залежностей забезпечують візуальні способи перевірки коду, аналізу залежностей між файлами.

Основним засобом розроблення у VisualStudio2012 є інтегроване середовище розроблення (IDE). IDE-середовище інтегроване із засобами модульного тестування та забезпечує можливості виявлення неефективного, небезпечного або погано написаного коду, керування змінами та модульне тестування як коду, так і бази даних.

Якість програмного продукту визначається за декількома критеріями: якісний програмний продукт повинен відповідати функційним та нефункційним вимогам, відповідно до яких він створювався, мати цінність для бізнесу, відповідати очікуванням користувачів.

У ЖЦ управління додатками якість повинна відстежуватись на всіх етапах ЖЦ ПЗ. Вона починає формуватись із визначення необхідних вимог. При заданні вимог необхідно вказувати бажану функційність та способи перевірки її досягнення.

Якісний програмний продукт повинен мати високу споживацьку якість, незалежно від галузі застосування: внутрішнє використання розробником, бізнес, наука та освіта, медицина, комерційні продажі, соціальна сфера, розваги та ін. Для користувача програмний продукт повинен задовольняти певному рівню його потреб.

Важливим аспектом створення якісного ПЗ є забезпечення нефункційних вимог, таких як зручність в експлуатації, надійність, продуктивність, захищеність, зручність супроводу. Надійність ПЗ визначає здатність без збоїв виконувати задані функції в заданих умовах протягом заданого відрізку часу. Продуктивність характеризується часом виконання заданих транзакцій або тривалих операцій. Захищеність визначає ступінь безпеки системи від пошкоджень, втрати, несанкціонованого доступу та злочинної діяльності. Зручність супроводу визначає легкість, з якою обслуговується продукт, в плані простоти виправлення дефектів, внесення коректив для відповідності новим вимогам, керування зміненим середовищем.

Управління ЖЦ програмного продукту допомагає розробникам цілеспрямовано добиватись створення якісного ПЗ, уникати втрат часу на переробку, повторне проектування та перепрограмування ПЗ.

Тестування програмного продукту дозволяє протягом всього ЖЦ ПЗ гарантувати, що програмні проекти відповідають заданим параметрам якості. Головна мета тестування - визначити відхилення в реалізації функцій них вимог, виявити помилки у виконання програм та виправити їх якомога раніше в процесі виконання проекту.

Важливим інструментом розробника ПЗ є модульне тестування, яке реалізується і середовищі иш^е8іРгатс^гк. Призначенням модульних тестів є перевірка того, що код працює вірно з точки зору програміста. Модульні тести формуються на більш низькому рівні, ніж інші види тестування, і перевіряють чи працюють функції, які лежать в їх основі, так, як очікується. Для модульного тестування використовується метод прозорої скриньки, для якого потрібне знання внутрішніх структур.

Модульні тести допомагають виявити проблеми проектування та реалізації. Крім того, модульний тест є гарною документацією з використання проектованої системи. Хоча модульне тестування вимагає додаткового програмування, але його застосування окупається за рахунок скорочення витрат на від лагодження додатку.

Модульні тести є важливим елементом регресійного тестування. Регресійне тестування представляє собою повторне тестування частини програми після внесення в неї змін або доповнень. Мета регресійного тестування - виявлення помилок, які можуть з'явитись при внесення змін до програми.

У VisualStudio є функція "Анализ покриття коду", яка проводить моніторинг того, які рядки коду виконувались під час модульного тестування. Результатом аналізу покриття коду є виявлення ділянок коду, які не покриті тестами.

Важливим аспектом створення якісного програмного продукту є дотримання розробниками правил та стандартів організації у написанні коду.

VisualStudio є функції аналізу коду, які дозволяють проаналізувати код, знайти типові помилки, порушення стандартів та запропонувати заходи з усунення помилок та порушень. Набори правил аналізу коду постачаються із VisualStudio. Розробники можуть налаштувати свої проекти на певний набір правил, а також додати свої специфічні правила аналізу коду.

В процесі аналізу коду використовуються метрики коду, які дають кількісні оцінки різних характеристик коду. Метрики дозволяють визначити складність коду та його ізольовані ділянки, які можуть призвести до проблем

при супроводі додатку. У VisualStudio використовуються наступні метрики коду:

- складність організації циклів - визначає кількість різних шляхів коду;
- глибина наслідування - визначає кількість рівнів в ієрархії наслідування об'єктів;
- об'єднання класів - визначає кількість класів, на які є посилання;
- рядки коду - визначає кількість рядків коду у виконуваному методі;
- індекс зручності підтримки - оцінює простоту обслуговування коду.

Для аналізу продуктивності та ефективності використання ресурсів додатковими інструментами у VisualStudio є інструменти профілювання. Профілювання представляє собою процес спостереження та запису показників про поведінку додатку. Інструменти профілювання (профілювальними) дозволяють виявити у додатку проблеми із продуктивністю. Такі проблеми, як правило, пов'язані із кодом, який виконується повільно, нефективно або надмірно використовує системну пам'ять. Профілювання, як правило, використовується для виялення ділянок коду, які під час виконання додатку виконуються часто або тривалий час.

Профілювальники бувають із вибіркою або з інструментуванням. Профілювальники із вибіркою роблять періодичні знімки виконуваного додатку та записують його стан. Профілювальники з інструментуванням додають маркери відстежування на початок та кінець кожної досліджуваної функції. В процесі роботи профілювальника маркери активізуються, коли потік виконання програми входить до досліджуваних функцій та виходить з них. Профілювальник записує дані про додаток і про те, які маркери були зачеплені під час виконання додатку.

Більшість корпоративних додатків працює із базами даних, що визначає необхідність розроблення та тестування додатків сумісно з базами даних командою проекту. У VisualStudio є інструментарій створення баз даних і розгортання змін у них. Для цього використовується автономне розроблення схем баз даних, яке дозволяє вносити зміни до схем без підключення до виробничої бази даних. Після внесення змін у середовище розроблення, VisualStudio дозволяє протестувати їх у самому середовищі та/або виділеному середовищі тестування. Крім того, VisualStudio дозволяє згенерувати псевдо реальні дані для проведення тестів. При позитивних результатах тестування VisualStudio дозволяє згенерувати сценарії для оновлення виробничої БД.

Цикл розроблення БД додатку складається із наступних кроків:

- переведення схеми БД у автономний режим;
- ітеративне розроблення додатку із БД;
- тестування схеми БД;
- побудова та розгортання БД та додатку.

Для вдосконалення процесу відлагодження додатків у VisualStudio є функція інтелектуального відстежування роботи програми IntelliTrace. Функція IntelliTrace конфігурується за допомогою наступних розділів:

- загальне (General) - включення/відключення функції, задання запису лише подій або додаткової інформації;
- додаткове (Advanced) - задання розташування журналу та його розміру для генерованого файлу;
- події IntelliTrace (IntelliTrace Events) - перераховуються всі події діагностики, які будуть збиратись під час відлагодження додатку;
- модулі (Modules) - список модулів, для яких необхідно збирати дані в процесі відлагодження додатку.

При записі подій додатку відбувається їх перехоплення при роботі додатку, інформація про події фіксується в журналі. При відлагодженні з IntelliTrace можна призупинити інтерактивний сеанс відлагодження та переглянути події або виклики. Також є можливість зупинки виконання додатку та покрокового руху назад та вперед в інтерактивному сеансі відлагодження, а також відтворення записаного сеансу відлагодження.

Тестування додатку є необхідним етапом управління ЖЦ додатку. Тестування виконує розробник в процесі створення коду додатку, а також тестувальник при перевірці якості розроблюваного програмного продукту. VisualStudio Ultimate надає розробнику інструмент для створення та використання модульних тестів (низькорівневні програмні тести, які дозволяють швидко перевірити наявність логічних помилок в методах класів), навантажувальних тестів (використовуються для дослідження роботоздатності додатку шляхом моделювання множини користувачів, які працюють з програмою одночасно) та веб-тестів продуктивності, а також тестів для інтерфейсу користувача (дозволяють автоматично сформувати код тесту, шляхом запису дій користувача при роботі із додатком, і потім виконувати ці тести автоматично).

Центральне місце у архітектурі засобів тестування VisualStudio посідає платформа модульного тестування, яка забезпечує підключення плагінів тестування (MS-TestManaged і MS-TestNative). Плагіни сторонніх розробників (NUnit, xUnit.net, MbUnit) можна підключити до платформи

юніт-тестування. Доступ до засобів тестування може здійснюватись з оглядача тестів, командного рядка та при побудові додатку через TeamBuild.

Для тестувальників у VisualStudio Ultimate є спеціалізований інструмент - MicrosoftTestManager, який дозволяє створювати плани тестування, формувати, додавати та видаляти тестові випадки, визначати та керувати фізичними та віртуальними тестовими середовищами, виконувати ручні та автоматичні тести.

Для тестувальників та розробників ПЗ VisualStudio включає диспетчер віртуального середовища LabManagement. Інструмент тестування LabManagement дозволяє створити інфраструктуру, яка максимально близько емулює реальне середовище планованого використання програмного продукту. Такі середовища можуть використовуватись для виконання автоматичних побудов, автоматизації тестів та виконання розроблюваного коду. Лабораторія тестування LabManagement інтегрована із середовищем TeamFoundationServer. Адміністрування віртуального середовища LabManagement проводить диспетчер MicrosoftSystemCenterVirtualMachineManager (SCVMM), за допомогою якого виконують необхідні налаштування віртуальної тестової лабораторії.

У VisualStudio та TeamFoundationServer додано засіб взаємодії із користувачами розроблених програмних продуктів, яке на запит дозволяє сформувати відгук користувача в БД для наступного опрацювання командою проекту.

Протягом всього ЖЦ ПЗ застосовуються різні типи тестування для гарантування того, що проміжні версії відповідають заданим показникам якості.

Якість коду визначається також і тим, наскільки важко або легко вносити зміни в код та наскільки код доступний для розуміння. Створений програмний додаток може виконувати необхідні функції, але мати проблеми із внесенням змін або розумінням створеного коду. В такому випадку ПЗ не можна назвати якісним, оскільки на етапі його супроводу можуть виникнути проблеми з його модифікацією при зміні вимог користувача. Для покращення якості програмного коду використовують рефакторинг - процес зміни ПЗ таким чином, що їїх зовнішня поведінка не змінюється, а внутрішня структура покращується.

Неякісний дизайн коду можна визначити за нступними ознаками:

- жорсткість - важке внесення змін в код;
- крихкість - пошкоджуваність програми в багатьох місцях із внесенням єдиної зміни;

- відсталість (інертність) - зусилля та ризики, пов'язані із спробою відокремити корисні для інших систем ділянки коду, надто великі;
- непотрібна складність - програма містить невикористовувані елементи;
- непотрібні повторення - повторення фрагментів коду у програмі;
- непрозорість - характеризує важкість коду для розуміння.

Для створення якісного дизайну коду доцільно використовувати деякі принципи та паттерни проектування ПЗ:

- принцип єдиного обов'язку - визначає, що в класу повинна бути лише єдина причина для зміни, тому класу доцільно доручати лише один обов'язок;
- принцип відкритості /закритості - визначає, що програмні сутності повинні бути відкритими для розширення, але закритими для модифікації;
- принцип підстановки - визначає, що повинна бути можливість замість базового класу підставити будь-який його підтип;
- принцип інверсії залежностей - визначає, що модулі верхнього рівня нее повинні залежати від модулів нижнього рівня (і ті, й інші повинні залежати від абстракцій), а абстракції не повинні залежати від деталей (деталі повинні залежати від абстракцій);
- принцип розділення інтерфейсів - визначає, що клієнти повинні знати лише про абстрактні інтерфейси, які мають властивість зчеплення;
- патерни проектування - пропонують універсальні, перевірені практикою рішення. Зі списку патернів слід виділити ті, які доцільно застосовувати при гнучкому розробленні ПЗ. Використання патернів дозволяє створювати ПЗ, яке легко модифікувати та супроводжувати.

4. Функціональні можливості та архітектура TeamFoundationServer (TFS)

Microsoft Visual Studio Team Foundation Server (TFS) призначений для забезпечення сумісної роботи колективів розробників ПЗ. Team Foundation Server надає наступні функційні можливості:

- управління проектами;
- відстежування робочих елементів;
- контроль за версіями;
- управління тестовими випадками;
- автоматизація побудови;
- звітність.

Архітектура Team Foundation Server є трирівневою сервіс-орієнтованою (рис. 4). Рівень додатку підтримується веб- сервеорм ASP.NET, розташованим в середовищі IIS. Рівень даних підтримується сервером баз даних MS SQLServer.

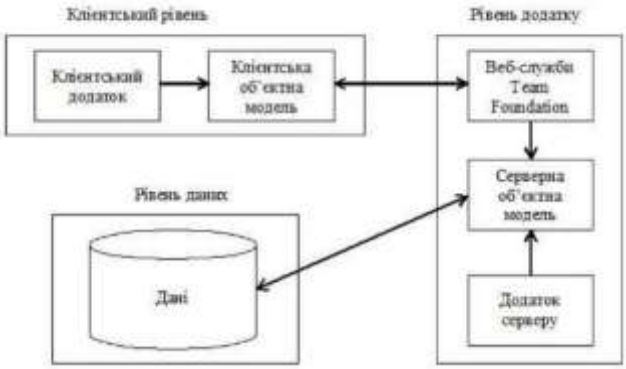


Рис.4 - Архітектура Team Foundation Server

Team Foundation Server - це, з логічної точки зору, веб- додаток, який складається з декількох веб-служб, які виконуються на рівні додатку. Дані служби реалізують функційність TFS. До складу веб-служб рівня додатку входять: управління версіями, служба побудови, відстежування робочих елементів, служби платформи TFS та лабораторії тестування LabManagement. Серверна об'єктна модель є інтерфейсом прикладного програмування для TFS. За необхідності розширення функційності TFS доцільно будувати на базі серверної об'єктної моделі.

Рівень даних складається з декількох реляційних баз даних та сховища даних: бази даних конфігурації серверу, бази даних аналітики, бази даних колекції командних проектів та сховища даних. Інформація командних проектів зберігається в реляційних базах даних, і через заплановані інтервали часу розташовується у сховище даних.

Клієнтський рівень може реалізовуватись в оболонці VisualStudio та у веб-броузері. Клієнтський рівень взаємодіє з рівнем додатків через серверну об'єктну модель та використовує веб-служби. Крім того, клієнтський рівень включає інтеграцію з Microsoft Office.

5. Способи розгортання TFS на одному або декількох серверах, в одному домені, робочі групі або в декількох доменах. Шаблони командних проектів TFS, області керування командними проектами

Для Team Foundation Server можна виконати розгортку декількома способами: на одному сервері; на декількох серверах; в одному домені, робочій групі або в декількох доменах.

У найпростішій серверній топології для розташування компонентів, які складають логічні рівні Team Foundation, використовується один фізичний сервер (рис. 5). При встановленні TFS з одним сервером всі компоненти (додаток TeamFoundationServer, SQLServer, ReportingServices і WidowsSharePointServices) встановлюються на одному комп'ютері. Така конфігурація передбачає виконання побудови (TeamFoundationBuild) та тестування або на сервері, або на клієнтських комп'ютерах. Загальна кількість користувачів для такої конфігурації, як правило, не більше 50.

У простій серверній топології для розташування компонентів, які складають логічні рівні Team Foundation, також використовується один фізичний сервер (рис. 5). Однак в такій топології враховується також додаткове навантаження на процесорні потужності, яке створюється програмним забезпеченням для побудови та тестування.

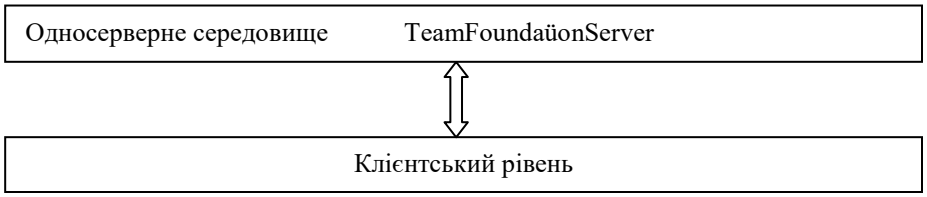


Рис. 5 - Найпростіша серверна топологія TFS

На рис. 6 веб-служби і бази даних для Team Foundation розташовуються на одному фізичному сервері, але служби побудови встановлюються на окремий комп'ютер. До Team Foundation Serverможна одержати доступ з клієнтських комп'ютерів, які належать до тієї ж робочої групи або до того ж домену. В даній конфігурації контролер побудови для виконання Team Foundation Build та контролер тестування встановлюються на окремих комп'ютерах. Загальна кількість користувачів для такої конфігурації, як правило, не більше 100.

Рис.6 - Проста серверна топологія TFS

У серверній топології середньої складності використовуються два або більше серверів для розташування логічних компонентів на рівні даних та додатків Team Foundation (рис. 7). На рис. 7 служби рівня додатків для Team Foundation Server розгортаються на одному сервері, а бази даних для TFS встановлюються на окремому сервері. На окремих серверах розташовуються веб-додаток SharePoint та екземпляр служб звітів SQL Server. Портал джля кожного командного проекту розташовується у веб-додатку SharePoint. Сервер Team Foundation Build та тестові контролери команди розгортаються на додаткових

Рис.7 - Серверна топологія TFS середньої складності

Для управління командним розробленням Team Foundation Server містить одну або декілька колекцій командних проектів, кожна з яких може містити ноль або більше проектів.

Командний проект представляє колекцію робочих елементів, коду, тестів та побудов, які охоплюють всі артефакти, використовувані у життєвому циклі програмного проекту. Командний проект будується на основі шаблону, який представляє набір XML-файлів, які містять деталі того, як повинен здійснюватись процес. У TFS 2012 є наступні шаблони проектів:

- MSF for CMMI Process Improvement 6.0, призначений для великих команд з суворо формальним підходом до управління проектами на основі моделі CMM/CMMI;
- MSF for Agile Software Development, який визначає гнучкий підхід до управління проектами розроблення ПЗ;
- Microsoft Visual Studio Scrum, призначений для невеликих команд (до 7-10 учасників), які використовують гнучку методологію та термінологію Scrum.

При створенні командного проекту надається можливість налагоджування та управління наступними областями проекту:

- відстежування робочих елементів дозволяє визначити початкові типи робочих елементів, загальні запити та запити користувача, створити початкові робочі елементи;
- класифікації дозволяють визначити початкові області та ітерації проекту;
- веб-сайт на базі SharePoint дозволяє керувати бібліотекою документів проекту;
- система контролю версій дозволяє визначити початкові групи безпеки, дозволи, правила повернення версій;
- звіти формуються у папки, створювані на сайті командного проекту;

групи та дозволи включають групи безпеки TFS та дозволи для кожної групи;

побудови включають стандартні процеси побудови для створення нових побудов;

- включає стандартні процеси лабораторії для використання, а також дозволи лабораторії для кожної групи;
- управління тестуванням включає стандартні конфігурації тестування, змінні, параметри та стани дозволів.

Робочими елементами у TeamFoundationServer є користувацький опис функційності, задачі, тестові випадки, помилки та перешкоди. Цими елементами потрібно керувати для виконання програмного проекту. Система відстежування робочих елементів дозволяє створювати робочі елементи, відстежувати їх стан, формувати історію їх зміни. Всі дані по робочих елементах зберігаються у базі даних TFS.

TeamFoundationServer включає централізовану систему контролю за версіями. Система контролю за версіями TFS надає наступні можливості:

- атомарні повернення - зміни, які вносяться у файли, запаковуються з «набором змін»; повернення файлу у наборі змін представляють собою неподільну транзакцію, чим гарантується узгодженість бази коду;
- асоціація операцій повернення з робочими елементами, що дозволяє відстежувати вимоги від початкової функції до задач та повернення у систему контролю за версіями коду;
- розгалуження та об'єднання при розробленні коду;
- набори відкладених змін дозволяють створювати копії коду, не записуючи їх у головне сховище системи контролю за версіями;
- використання підписів дозволяє позначити набір файлів у певній версії за допомогою текстового підпису;

- одночасне видобування дозволяє декільком членам команди одночасно редагувати файл з наступним об'єднанням змін;
- відстежування історії файлів;
- політики повернення, які надають можливість виконати код для перевірки припустимості повернення;
- примітки при поверненні дозволяють формувати метадані про повернення;
- проксі-сервер дозволяє оптимізувати роботу з регіональними центрами розроблення.

Побудова проекту програмного продукту у TFS реалізує сервер Team Foundation Build. Він дозволяє стандартизувати інфраструктуру побудов для команди проекту. Побудова проекту може виконуватись у наступних режимах:

- ручний, при якому побудова вручну ставиться у чергу побудов;
- неперервна інтеграція, яка дозволяє виконувати побудову при кожному поверненні в систему контролю версій;
- побудов, при якому повернення групуються разом, щоб в побудову включались зміни за певний період часу;
- повернення, при якому поверненн приймаються лише в тому випадку, якщо успішно пройшли злиття та побудови відправлених змін;

розклад, який дозволяє налагодити час побудови на певні дні тижня.

Розробник має можливість взаємодіяти з ключовими службами TeamFoundationServer через:

- командний оглядач (TeamExplorer) VisualStudio 2012, який надає доступ до функцій керування життєвим циклом додатків, включаючи командні проекти, аналіз коду, керування версіями та побудовами;
- доступ через веб (Team Web Access), завдяки якому надається веб-доступ до функцій керування ЖЦ додатків, включаючи командні проекти, робочі групи, керування проектами, керування версіями та побудовами;
- MicrosoftExcel, завдяки якому надається можливість визначати та змінювати обочі елементами масивом, а також створювати звіти на основі запиту робочого елемента;
- MicrosoftProject, завдяки якому надається можливість керувати планом проекту, задачами розкладу, ресурсами, формувати календар проекту, діаграми Ганта та представлення ресурсів;
- консоль адміністрування Team Foundation Server, завдяки якій здійснюється налагодження TFS;
- інструменти командного рядка;

сторонні засоби інтеграції.

6. Питання створення командного проекту, зміст програмної інфраструктури проекту, склад і призначення робочих елементів

При командному розробленні ПЗ важливими питаннями є планування робіт, складання розкладу, керування областю проекту, комунікації, складання звітів, аналіз та постійне вдосконалення процесу. Для вирішення цих питань TeamFoundationServer пропонує наступні інструменти:

- шаблон процесу, який визначає процес, використовуваний командним проектом;
- керівництво з процесу, яке містить опис шаблонів процесу;
- колекція командних проектів, яка представляє собою контейнер для декількох командних проектів;
- командний проект, який зберігає та організовує дані про весь життєвий цикл розроблення ПЗ;
- портал проекту/панелі моніторингу, на яких надається інформація по проекту для всіх членів колективу;
- елементи планування для управління списками вимог як на рівні проекту, так і на рівні ітерації;
- відстежування робочих елементів, яке дозволяє відстежувати стан робочих елементів та іншу інформацію, пов'язану з ними;
- звітність, яка дозволяє формувати звіти під час ЖЦ проекту;
- інтеграцію з MicrosoftProject та MicrosoftExcel для управління проектами з середовища MicrosoftOffice.

Розроблення ПЗ починається із створення командного проекту. Командний проект містить інформацію про кожен крок ЖЦ розроблення ПЗ, включаючи вимоги користувачів, тестові випадки, помилки, перешкоди, побудови.

При створенні командного проекту необхідно задаим його ім'я, опис, визначитись із шаблоном процесу, вимогами до системи контролю за версіями та необхідністю створення порталу для проекту.

В результаті створення командного проекту Team Foundation Server формує наступні папки:

- Моя работа - папка, в якій задаються виконувана та призупинена роботи, доступні робочі елементи та активні запитані задачі аналізу коду;
- Ожидающие изменения - папка, в якій забезпечується можливість збереження змін коду в базі даних TFS та видобування проектних рішень для редагування користувачем;

- Рабочие элементы проекта - папка, в якій зберігаються згруповані дані про поточні робочі елементи проекту;
- Построения - папка, в якій формуються та зберігаються визначення та результати побудови додатків;
- Отчеты - папка, в якій є підпапки із звітами та посилання на стандартні звіти;
- Документы - папка, в якій зберігаються документи проекту;
- Параметры - папка, в якій зберігаються посилання для переходу до вікон задання параметрів проекту (безпека, склад груп, система управління версіями, області та ітерації робочих елементів, параметри порталу та оповіщення проекту).

Після створення командного проекту керівник формує колектив (команду). Для кожного члена команди керівник проекту визначає доступ до проекту в цілому та до окремих артефактів, важливих для роботи кожного члена команди.

Для структурування проекту використовуються області та ітерації. Області можуть визначатись, виходячи з певного функційного призначення етапу робіт, а ітерації - у вигляді набору робіт на заданому часовому інтервалі.

При плануванні командного проекту ключовими сутностями є робочі елементи. Залежно від шаблону командного проекту набір та найменування робочих елементів дещо відрізняються. Робочий елемент - запис, створений у Visual Studio Team Foundation Server для задання визначення, призначення, пріоритету та стану елементу роботи. Робочі елементи можуть включати найменування, опис призначення, стан, кому призначено, цінність для бізнесу, приналежність до робочої області.

Так, для гнучкої методології Agile робочими елементами є:

- Користувацький опис функційності (UserStory) - вимога користувача, яку необхідно виконати при реалізації проекту; такі робочі елементи можуть бути пов'язані між собою, а також з дочірніми (Задача, Помилка) або батьківськими елементами.
- Задача (Task) - створюється в проекті для призначення та виконання роботи, надає деталі реалізації для вимог користувача;
- Помилка (Bug) - використовується для відстежування та моніторингу роблем у програмному продукті;
- Перешкода (Issue) - використовується дл фіксації у проекті подій або об'єктів, які створюють проблеми у виконанні проекту та повинні бути усунені під час поточної або майбутньої ітерації;

- Тестовий випадок (TestCase) - описує умови перевірки вірності виконання програмним продуктом вимог користувача;
- Невиконана робота - використовується для запису вимог користувача при плануванні командного проекту.
- Розроблення програмного коду. Задачі призначаються розробникам програмного коду. Розробники проводять кодування та модульне тестування задач у середовищі Visual Studio.

Розроблювані коди знаходяться під контролем системи керування версіями у сховищі коду. Роботоздатні коди задач збираються в побудові за допомогою Team Service Build і передаються у систему керування версіями на Team Foundation Server. В процесі роботи розробник проводить видобування (checkingout) файлів з кодами задач додатку в свою робочу область на інструментальному комп'ютері. Після виконання певного етапу робіт розробник здійснює повернення (checkingin) у сховище TFS коду. При поверненні коду формується набір змін (changeset), який мітсить всю інформацію, пов'язану із поверненням (посилання на робочі елементи, виправлення, примітки, політики та дані про власника, дату та час). Це дає можливість розробникам переглядати різні версії файлів та аналізувати внесені зміни.

Тестування. Тестувальники на основі тестових випадків готують тести для заданих вимог користувача. Для тестування файли коду видобуваються зі сховища TFS і підлягають тестуванню. У випадку виявлення помилки формується робочий елемент Помилка, який спрямовується розробнику для виправлення. Якщо тести проходять, то відповідна вимогам користувача вважається виконаною та для неї встановлюють стан «Готово».

Звіти. Team Foundation Server має потужну систему збирання інформації під час проекту та підготовки звітів. Звіти призначеня як для керівників проекту, так і для членів команди. Система звітності базується на сховищі даних TFS. Дані можуть бути представлені у вигляді звітів, які дозволяють переглядати метрики проекту. Система звітів дозволяє відстежувати робочі елементи, побудови, статистику системи контролю за версіями, результати тестів, індикатори якості проекту. Team Foundation Server надає набір звітів, адле є можливість створювати також і звіти користувача.

У TFS є три схвоища даних:

- операційне сховище;
- сховище даних;
- OLAP-куб.

Операційне сховище представляє собою набір реляційних баз даних для зберігання такої інформації, як сирцевий код, звіти побудови, результати тестів та відстежування робочих елементів.

Схвоище даних призначене для виконання запитів та створення звітів. Сховище даних одержує дані з операційного сховища через певні проміжки часу. Сховище даних концептуально побудоване за схемою «зірка».

OLAP-куб TeamFoundationServer є багатовимірною базою даних, яка містить агреговані дані для підготовки аналітичних звітів. OLAP-куб одержує дані зі сховища даних через заплановані інтервали часу. Дані багатовимірної бази даних можуть використовуватись різними клієнтськими додатками, включаючи Microsoft Excel і конструктор SQL-звітів.

- Team Foundation Server включає два набори звітів:
- звіти Microsoft Excel Reports
- звіти служби звітності SQL Reporting Services Reports.

MicrosoftExcel дозволяє створювати звіти з OLAP-кубу TFS

або використовуючи запити робочих елементів проекту. Звіти можуть бути опубліковані на SharePoint-порталі проекту.

7. Аналіз методології Scrum, робочі елементи шаблону Microsoft Visual Studio Scrum

Методологія Scrum представляє собою ітеративний процес розроблення ПЗ. При такому розробленні для програмного продукту створюється багато послідовних випусків, в яких поступово додається потрібна функційність. Ітеративний підхід дозволяє по завершенню поточної ітерації продемонструвати замовнику роботоздатний програмний продукт, можливо з обмеженою функційністю, одержати відгук, зауваження та додаткові вимоги, які будуть варховані у наступних ітераціях. Основними артефактами в методології Scrum є робочі елементи, звіти, книги та панелі моніторингу.

Робочі елементи використовуються для відстежування, спостереження за станом ходу розроблення ПЗ та створення звітів. Робочий елемент - це запис, який створюється у Visual Studio Team Foundation Server для задання визначення, призначення, пріоритету та стану елементу роботи. Для шаблону Microsoft Visual Studio Scrum визначаються наступні типи робочих елементів:

- Невиконана робота;
- Помилка;
- Задача;
- Перешкода;
- Тестовий випадок.

В методології Scrum вимоги користувача, які визначають функційність продукту, задаються елементами заділу роботи продукту (Product Backlog Item - PBI). Елементи заділу роботи продукту, які називають також елементами роботи (ЕР), представляють собою короткий опис функцій продукту та оформляються у довільній формі у вигляді коротких приміток. Спочатку задаються найбільш важливі та зрозумілі всім вимоги користувача. ЕР можуть деталізуватись у вигляді задач. В процесі створення програмного продукту ЕР можуть уточнюватись, додаватись або видалятись зі списку вимог.

Цикл випуску продукту в Scrum складається з множини ітерацій, які називаються спринтами. Спринт має фіксовану тривалість, як правило, 1-4 тижні. Елементи роботи, включені до чергового спринту, не підлягають зміні до його завершення. Хоча спринт завершується підготовкою роботоздатного програмного продукту, його поточної функційності може бути недостатньо для оформлення випуску, який має цінність для замовника. Тому випуск роботоздатного програмного продукту, який замовник може використовувати, включає, як правило, декілька спринтів.

8. Організація колективу у методології Scrum. Життєвий цикл проекту

Організація команди в методології Scrum визначає три ролі:

- власник продукту (Product owner);
- керівник (ScrumMaster);
- члени команди (Team members).

Власник продукту відповідає за все, що пов'язано зі споживацькими якостями програмного продукту. Він визначає вимоги користувача, аналізує їх реалізацію, має право змін вимог, контролює якість продукту. Він може бути представником замовника в команді або членом команди розробників, який представляє інтереси замовника. Власник продукту виконує наступні основні задачі:

- визначення та пріоритезація вимог/функцій, тобто елементів робіт та задач;
- планування спринтів та випусків;
- тестування вимог/функцій.

Керівник відповідає за стан та координацію проекту, продуктивність команди та усунення перешкод, які заважають проекту. В обов'язки керівника входить:

- проведення щоденних Scrum-зборів;
- залучення співробітників поза командою;
- стимулювання ефективного спілкування членів команди;
- визначення розміру команди.

Члени команди відповідає за розроблення програмного продукту високої якості. Вони повинні володіти навичками в проектуванні та архітектурі програмного продукту, бізнес-аналізі, програмуванні, тестуванні, налагодженні баз даних та проектуванні інтерфейсу користувача. Члени команди приймають участь у

плануванні спринтів. Команда може включати досвідчених розробників та новачків, які в процесі роботи повинні вдосконалюватись при обміні знаннями з іншими членами команди. Члени команди відповідають за наступні задачі у проекті:

- обов'язкове виконання елементів робіт, включених в поточний спринт;
- акцент на взаємозв'язаних задачах спринту;
- вдосконалення команди.

Інструментальна та методична підтримка гнучкого підходу до створення програмних продуктів Scrum, реалізована у VisualStudio, дозволяє керувати життєвим циклом проекту ПЗ (рис. 8).

На початку проектування власник продукту та замовник формують концепцію програмного продукту, яка відображає, для кого призначено продукт, які переваги отримають користувачі та які існують конкуренти. Концепція продукту пов'язується з галуззю проекту та обмеженнями. Галузь проекту визначає масштаб робіт, а обмеження - умови, якими керуватимуться для перших спринтів та випусків. Далі власник продукту створює список всіх потенційних функцій продукту - «Невиконана робота по продукту» (ProductBacklog). Невиконана робота по продукту, яку надалі називатимемо невиконана робота - НВР, містить список елементів роботи - користувацьких описів функційності.

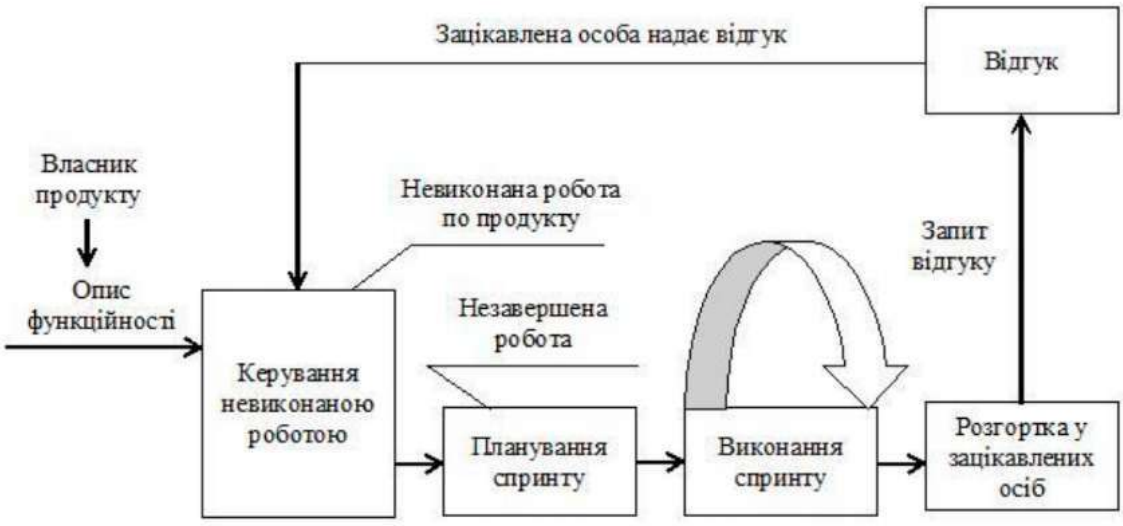


Рис. 8 - Життєвий цикл проекту ПЗ

Керування невиконаною роботою по проекту зводиться до підтримки елементів робіт в актуальному стані. Окремі елементи робіт списку НВР можуть додаватись або вилучатись в процесі створення ПЗ. Це є результатом того, що колектив отримує додаткову інформацію про нові вимоги замовника до проектованого програмного продукту, а замовник з'ясовує, як реалізуються його очікування.

Для елементів робіт (ЕР) власник продукту сумісно з командою проекту визначає пріоритети. При планування спринту в нього включають найбільш значущі, з точки зору власника продукту, вимоги користувача, які характеризуються найбільшою споживацькою цінністю. Обрані елементи робіт переміщають у список «Незавершена робота» (SprintBacklog). Список «Незавершена робота» (НЗР) відбиває склад робіт планованого спринту. Список НЗР є результатом процесу планування спринту.

Координацією робіт у спринті займається керівник спринту (ScrumMaster). Він організовує процес пріоритезації задач спринту, розподілу задач між членами команди. Керівник спринту проводить збори з планування робіт, щоденні збори для короткого обговорення результатів роботи та проблем, оглядові збори в кінці спринту та випуску.

Щоденні Scrum-збори мають тривалість 15-30 хвилин. Метою таких зборів є виявлення проблем, які гальмують процес розроблення, і визначити дії з їх нейтралізації. Для простих проблем приймається рішення щодо їх усунення, а складні проблеми відкладаються на наступні спринти. Протягом щоденних Scrum-зборів керівник задає темп спринту, акцентує команду на найбільш важливих елементах списку невиконаних робіт. Кожний член колективу повідомляє, що було зроблено вчора, що буде робити сьогодні і які перешкоди є у роботі. Якщо на щоденних Scrum-зборах виникають питання, для вирішення яких необхідні фахівці, яких в колективі немає, тоді керівник бере на себе аналіз та можливі шляхи вирішення даного питання.

Результатом спринту є роботоздатне ПЗ, яке можливо володіє лише частиною необхідних функцій програмного продукту. Випуск спринту може бути розгорнутий у зацікавлених осіб для попереднього аналізу відповідності очікуванням замовника. Результатом відгуку є формування «Відгук» (FeedBack), що може призвести до змін вмісту списку «Невиконана робота по продукту». Після виконання всіх робіт за програмним продуктом, тобто обнуління списків вимог «Невиконана робота по продукту» та «Незавершена робота», готується фінальний випуск програмного продукту.

Список «Невиконана робота по продукту» є одним з ключових артефактів у методології Scrum. Успіх Scrum-команди багато в чому визначається якісним вмістом даного списку. Список НВР включає

користувацькі описи функційності - елементи роботи, а також може включати нефункційні вимоги. Для створення списку НВР можуть застосовуватись різні клієнтські сервіси (рис.9).

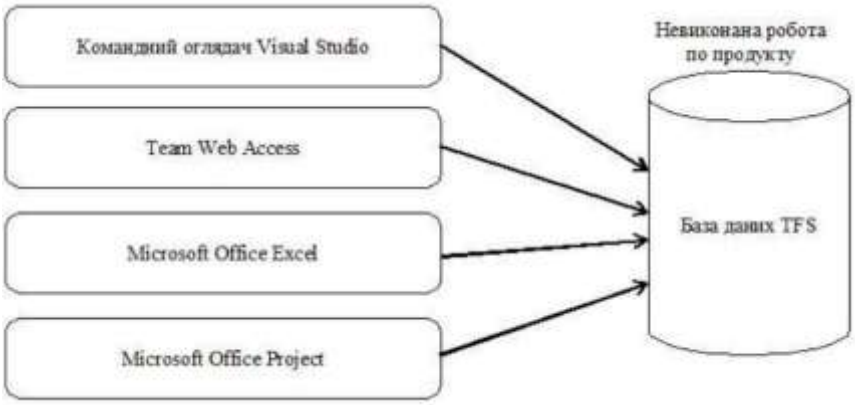


Рис.9 - Клієнтські сервіси TFS

Власник продукту на основі вимог та побажань клієнтів формує список функцій продукту у вигляді робочих елементів продукту, які розташовує у список «Невиконана робота по продукту». При створенні нового елементу невиконаної роботи для нього встановлюється стан «Новий» (рис.10).

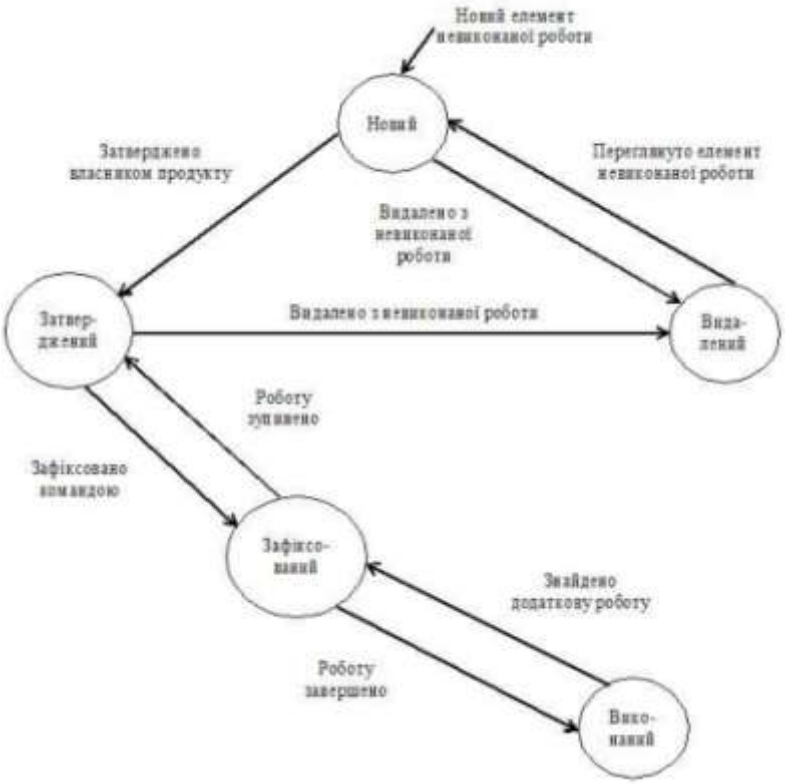


Рис.10 - Робочий процес елементу невиконаної роботи

Після встановлення елементу роботи пріоритету його стан змінюють на «Затверджений». На зборах з планування спринту команда переглядає найбільш пріоритетні елементи роботи і обирає ті, які будуть виконуватись у поточному спринті. Для елементів невиконаної роботи, які потрапили у поточний спринт, встановлюється стан «Зафіксований». Це означає той факт, що робочі елементи спринту не підлягають зміні до кінця спринту. При завершенні роботи за елементом його стан встановлюється як «Виконаний». Якщо для елементу роботи, який знаходиться у стані «Виконаний», виявляється додаткова робота, то цей елемент може бути переведний у стан «Зафіксований». Для елементу роботи, який знаходиться у стані «Зафіксований», при виникненні проблем, які перешкоджають його завершенню у спринті, робота може бути призупинена і встановлено стан «Затверджений». Елемент невиконаної роботи може бути видалений зі списку «Невиконана робота по продукту» за рішенням власника продукту. Це може відбутись як зі стану «Новий», так і зі стану «Затверджений». Для видаленого елемента встановлюється стан «Видалений». В результаті перегляду елементу роботи зі станом «Видалений» для нього знов можливе переведення у стан «Новий».

Список «Невиконана робота по продукту» є головним документом для Scrum-команди. На основі даного списоу команда створює інші робочі елементи, які скаладають спринти й випуски. Для елементів невиконаної роботи команда створює задачі та тестові випадки. Задачі деталізують елемент роботи та визначають конкретну реалізацію вимог користувача. Тестові випадки необхідні для перевірки відповідності функційності коду вимогам користувача. Якщо тестовий випадок не проходить, то створюється робочий елемент «Помилка». При блокуванні задачі через неможливість її виконання в поточному спринті створюють робочий елемент «Перешкода». Scrum-команда може створювати допоміжні робочі елементи по відношенню до елементів, на які вони впливають (задачі та тестові випадки), та пов'язувати ці елементи (рис.11).

Для відстежування ходу виконання проекту, можна створювати звіти, які відображають найбільш важливі дані для поточного проекту. В процесі створення ПЗ можна користуватись стандартними звітами або створювати власні звіти. Звіти можна створювати, налагоджувати і перевіряти за допомогою MS Excel, MS Project або служб Reporting ServicesSQL Server.

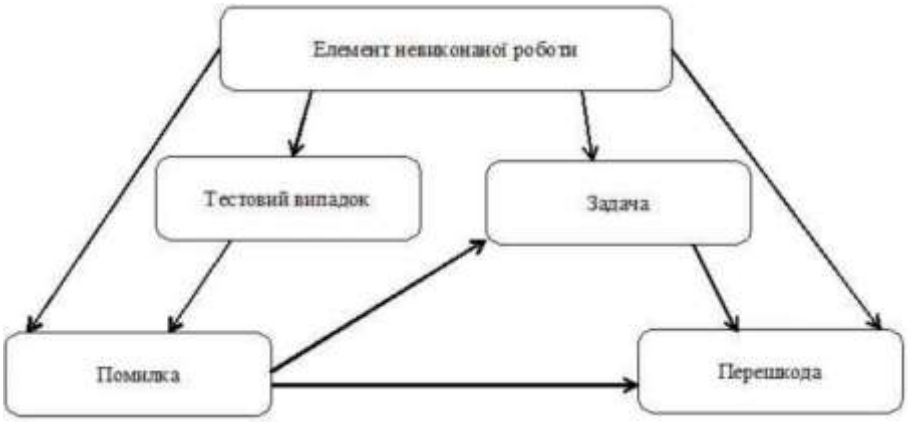


Рис.11 - Зв'язок між робочими елементами

Методологія Scrum має наступні позитивні сторони:

- користувач іпочинають бачити систему вже через декілька тижнів і можуть виявляти проблеми на ранніх стадіях розроблення програмного продукту;
- інтеграція технічних компонентів відбувається під час кожного спринту і тому пробоеми проекту (якщо вони виникають) виявляються практично одразу;
- в кожному спринті команда фокусується на контролюванні якості;
- гнучка робота із змінами в проекті на рівні спринту.

Microsoft Solutions Framework є методологією розроблення ПЗ, яка представляє собою узагальнення кращих проектних практик, які використовувались командами розробників Microsoft. Методологія MSF є складовою частиною продукту Visual Studio Team System. В основі методології MSF лежить ітеративний інтегрований підхід до створення та впровадження IT-рішень, який базується на фазах та віхах. В моделі команди MSF відсутній офіційний лідер, всі відповідають за проект в рівній мірі. В залежності від розміру та складності проекту модель команд MSF припускає масштабування.

Гнучка методологія розроблення ПЗ орієнтована на використання ітеративного підходу, при якому програмний продукт створюється поступово, за декілька ітерацій, які включають реалізацію певного набору вимог. Результатом ітерації є проміжний варіант роботоздатного ПЗ.

Visual Studio і Team Foundation Server прдставляють рішення компанії Microsoft щодо керування життєвим циклом додатків. Керування життєвим циклом у Visual Studio базується на принципах продуктивності, інтеграції та можливості розширення. У Visual Studio для архітектурного проектування використовуються інструменти візуального моделювання на

основі мови UML. Team Foundation Server призначений для забезпечення сумісної роботи команд розробників ПЗ та має трирівневу сервіс-орієнтовану архітектуру.

Методологія Scrum представляє собою ітеративний процес розроблення ПЗ. Робочі елементи використовуються для відстежування, спостереження за станом ходу розроблення ПЗ та створення звітів. Організація команди в методології Scrum визначає ролі власника продукту, керівника та членів команди. Життєвий цикл проекту ПЗ включає формування та керування невиконаною роботою, планування та виконання спринту, розгортку ПЗ у зацікавлених осіб, одержання відгуків для покращення програмного продукту. При керуванні робочим процесом постійно відстежуються зв'язки та змінюються стани елементів невиконаної роботи по продукту.

https://learn.microsoft.com/ru-ru/azure/devops/server/download/azuredevopsserver?source=recommendations&view=azure-devops

https://learn.microsoft.com/ru-ru/azure/devops/server/install/get-started?view=azure-devops-2022&viewFallbackFrom=azure-devops#installations-for-evaluation-or-personal-use

Рекомендована література

1. Соммервилл. Инженерия программного обеспечения. 6-е издание. - Издательский дом "Вильямс", 2002.
2. Ф.А.Новиков, Э.А.Опалева, Е.О.Степанов. Учебно- методическое пособие по дисциплине «Управление проектами и разработкой ПО» // [Електронний ресурс] - Режим доступу: http://window.edu.ru/resource/366/6G366/files/itmo3G5.pdf.
3. Технологии командной разработки программного обеспечения информационных систем // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/4806/1054/lecture/9998.
4. Д.Андреев. Организация процессов разработки программного обеспечения с использованием Team Foundation Server 2010 // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/649/505/info.

5. Бьерк А., ДелаМазаМ., Резник С. ScrumTeamFoundationServer 2010. Профессиональный подход. - М. ЭКОМ Паблишерз, 2012.

6. Левинсон Дж. Тестирование ПО с помощью Visual Studio 2010. - М.: ЭКОМ-Паблишерз, 2012.

7. Шаблоны и средства Microsoft — Microsoft Visual Studio 2010 // [Електронний ресурс] - Режим доступу: http://msdn.microsoft.com/ru-ru/vstudio//aa718795.aspx

8. Шаблон процесса гибкой разработки для Visual Studio ALM // [Електронний ресурс] - Режим доступу: http://msdn.microsoft.com/ru-ru/library/dd380647.aspx

9. Шаблон процесса Scrum для Visual Studio ALM // [Електронний ресурс] - Режим доступу: http://msdn.microsoft.com/ru-ru/library/ff731587.aspx

10. Управления жизненным циклом приложений с помощью Visual Studio и Team Foundation Server // [Електронний ресурс] - Режим доступу: http://msdn.microsoft.com/ru- ru/library/fda2bad5.aspx

11. Мейер Дж. Д. Командная разработка с использованием Visual Studio Team Foundation Server / Дж. Д.Мейер, Дж. Тейлор, А. Макман, П. Бансод, К. Джонс - Изд. Корпорация Microsoft, 2007.

Лабораторна робота №10

«Дослідження процесу розробки документації на програмну продукцію»

Мета роботи: засвоєння базових знань щодо розробки документів, які входять до складу пакету документації на програмну продукцію; отримання навичок формування основних документів із пакету програмної документації.

Підготовка до роботи: Вивчити і уяснити призначення і зміст,завдання, інструментальні засоби та основні процеси розробци документації на програмну продукцію (ПД).

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Ознайомитись з діючими стандартами.
4. Для проведення досліджень встановити і налагодити технологічне програмне забезпечення в обчислювальному середовищі віртуальних машин.
5. Дослідити методології розроблення ПД на основі стандартів серії ГОСТ 34 і керівного документу РД 50-34.698-90.
6. Дослідити основні принципи використання програмних засобів Doxygen, Fpydoc, Jsdoc, PhpDocumentor, Sancfstle, Sphinx.
7. Дослідити процес створення документації на ПД на прикладі раніш виконаної курсової або самостійної роботи.
8. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

9 Назва та мета роботи.

2. Коротко теоретичні відомості

11. Завдання дослідження лабораторної роботи згідно до обраного варіанта.
12. Результати виконаного дослідження.
13. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Тематика індивідуального домашнього завдання пов'язана зі створенням пакету документації на програмний засіб, розроблений у рамках будь-якої виконаної курсової роботи (курсового проекту). При документуванні програмного засобу слід керуватися стандартами документування, наприклад, ГОСТ 34.201-89, ГОСТ 34.602-89.

Будь-які стандарти на організацію життєвого циклу систем та програмної продукції вводять процеси (стадії, етапи) щодо документування готового програмного продукту та процедур його розробки. Крім того, реалізація в компанії процесів документування ходу розробки програмної продукції свідчить про високий рівень зрілості цієї компанії з точки зору здатності розробляти якісний продукт.

Процеси документування програмної продукції (ПП), пов'язані зі знаннями, навичками та вміннями документувати процеси життєвого циклу ПП, визначені у професійних стандартах за низкою сфер діяльності (технічний письменник, програміст та ін.).

*Документація на ПП (*автоматизовану систему) це комплекс взаємопов'язаних документів, у якому повністю описані всі рішення щодо створення та функціонування продукту, а також документів, що підтверджують відповідність продукту вимогам завдання на виконання робіт (технічного завдання) та його готовність до функціонування в реальному середовищі.

Відповідно до стандартів серії ГОСТ 34 основними документами, що випускаються в рамках життєвого циклу програмної продукції, є:

- технічне завдання розробку;
- опис функцій, що автоматизуються;
- опис постановки завдань чи їхнього комплексу;
- опис інформаційного забезпечення системи;
- опис організації інформаційної бази;
- опис систем класифікації та кодування (рекомендується використовувати Єдину систему класифікації та кодування техніко-економічної та соціальної інформації (єдина система конструкторської документації ЄСКД);
- опис комплексу технічних засобів;
- опис програмного забезпечення;
- опис організаційної структури;
- проектна оцінка надійності системи;
- специфікація обладнання;

- каталог бази даних;
- методика автоматизованого проектування;
- керівництво користувача;
- програма та методика випробувань системи.

Вимоги до змісту зазначених документів встановлені у керівному документі зі стандартизації РД 50-34.698-90.

Документи, що належать до одного комплекту документації на програмний засіб, упорядковуються та кодуються відповідно до наступної структури (рис. 1)

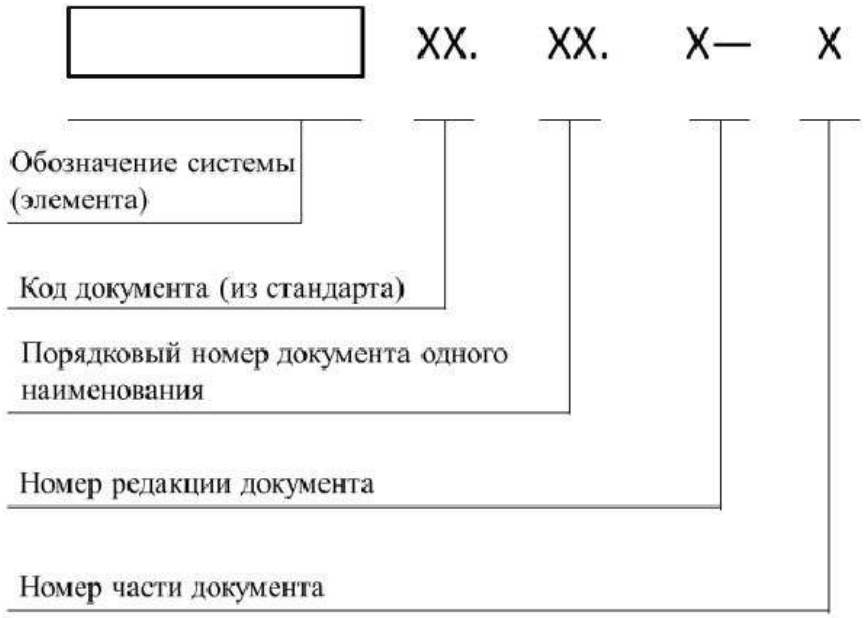
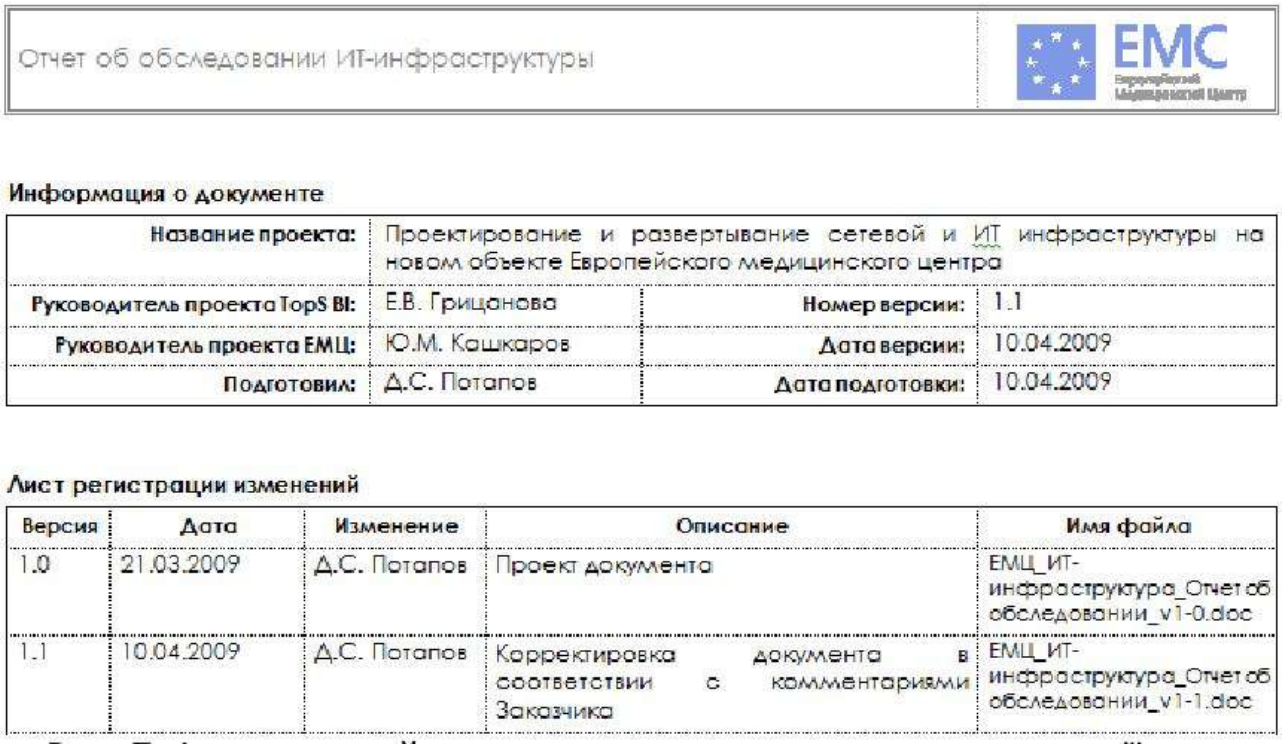


Рис. 1. Позначення документів із пакета документації на програмну продукцію

Адаптоване позначення документа, що включає аркуш реєстрації змін, може мати вигляд, представлений на рис. 2.

Вимоги до документування програмної продукції згідно з іншим стандартом - ГОСТ Р ИСО/МЭК 12207-2010 - позначені в рамках одного із процесів ЖЦ, а саме процесу менеджменту програмної документації.

Для автоматизації процесу документування програмної продукції на основі спеціальним чином оформлених коментарів, що містяться у вихідному коді, використовуються генератори документації (для різних мов та середовищ програмування). До них належать:Doxygen(C++, C, Objective C, C#, Java, PHP), *Epydoc(Python), Javadoc(Java),JSDoc(Java Script),PhpDocumentor(PHP), Sandcastle(C#,* сімейство мов платформи .NET), *Sphinx(Python)* та ін.



Отчет об обследовании ИТ-инфраструктуры

EMC

Информация о документе

Название проекта:	Проектирование и развертывание сетевой и ИТ инфраструктуры на новом объекте Европейского медицинского центра		
Руководитель проекта TopS BI:	Е.В. Грицанова	**Номер версии:**	1.1
Руководитель проекта ЕМЦ:	Ю.М. Кашкаров	**Дата версии:**	10.04.2009
Подготовил:	Д.С. Потапов	**Дата подготовки:**	10.04.2009

Лист регистрации изменений

Версия	Дата	Изменение	Описание	Имя файла
1.0	21.03.2009	Д.С. Потапов	Проект документа	ЕМЦ_ИТ-инфраструктура_Отчет об обследовании_v1-0.doc
1.1	10.04.2009	Д.С. Потапов	Корректировка документа в соответствии с комментариями Заказчика	ЕМЦ_ИТ-инфраструктура_Отчет об обследовании_v1-1.doc

Рис. 2. Адаптований опис документу та пакету документації

Стандарти документування покривають такі типи документації на програмний засіб:

- проектна (включає текстові описи, діаграми, моделі програмного засобу, що розробляється на основі методологій проектування IDEF0, DFD, IDEF3, IDEFX, UML та ін.);
- технічна (використовується для визначення та опису структур даних та алгоритмів; надається з вихідним кодом програмної продукції або включається до нього);
- користувальницька (орієнтована на кінцевого користувача).

Крім цього виділяють четвертий тип документації - маркетингову документацію, що включає рекламні матеріали, наприклад бізнес-план з просування програмної продукції, графічні матеріали з оформлення коробки продукту і т.п.

Індивізуальні практичні завдання

1. Ознайомтеся з пакетом документів реального проекту (Додаток А).

2. За прикладом документації реального проекту та стандартів серії ГОСТ 34 розробіть пакет документів на програмну продукцію, отриману в рамках виконання будь-якої курсової/самостійної роботи або курсового проекту. Пакет документів обов'язково має включати технічне завдання на

створення, програму та методику випробувань, інструкцію (посібник) щодо використання. Інші види документів включаються до пакету за рішенням розробника (студента).

Додаток А. Приклад оформлення програмної документації

Додаток №1
до договору №
_____
від ”___” _____ 2xxx
року

ПОГОДЖЕНО
_____
_____ _____

”____” _____ 2xxx р.

ЗАТВЕРДЖЕНО
_____
_____ _____

”____” _____ 2xxx р.

ТЕХНІЧНЕ ЗАВДАННЯ
на створення науково-технічної продукції

за договором № _____ від "___" _____ 2xxx року

20XX_

1. Назва роботи

"Розроблення програми для …….".

2.Підстави для виконання

Кошти …….

3. Перелік скорочень

АС	Автоматизована система
БД	База даних
КЗЗ	Комплекс заходів захисту інформації
КСЗІ	Комплексна система захисту інформації
……	……..
НДР	Науково-дослідна робота
НСД	Несанкціонований доступ
ПЗ	Програмне забезпечення
СУБД	Система управління базами даних
СЗІ	Система захисту інформації

4. Мета та актуальність роботи

Метою НДР є підвищення оперативності та якості інформаційно-аналітичної діяльності …….. шляхами:

- розроблення архітектурно-топологічного рішення щодо створення багаторівневої технології збирання, накопичення та надання інформації щодо ……..;
- уніфікації та стандартизації форм електронних документів щодо …….;
- створення стандартних типових форм даних на базі технології стандарту …….;

- створення програмних засобів інформаційної взаємодії користувачів та поставщиків інформації в базу даних ……;
- впровадження програмних засобів інформаційної взаємодії користувачів та поставщиків інформації в базу даних ……;
- створення СЗІ для інформаційної технології ведення бази даних ……

4.1. В ході виконання НДР мають бути вирішені наступні основні задачі:

- Формування видачі …….на стадії обговорення.
- ……

4.2. Очікувані результати НДР

На основі досліджень, виконаних в рамках даної НДР, повинні бути розроблені алгоритми, БД, СЗІ АС та ПЗ ведення БД ……. та створено макет вищезазначеної АС та ПЗ.

Результатом виконання робіт є:

- створення технології формування видачі …….. на стадії обговорення;
- ……..

4.3. Нормативно-правова база, з урахуванням якої повинна виконуватись НДР

Науково-дослідна робота проводиться з урахуванням вимог наступних нормативно-правових документів:

- Закону України „…….”;
- Закону України „Про захист інформації в автоматизованих системах”;
- Постанови КМ України № 302 від 13.03.2002 р. „……..”;
- ДСТУ 3008-95 „Документація. Звіти у сфері науки і техніки. Структура і правила оформлення”;
- ДСТУ 3973-2000 „Правила виконання науково-дослідних робіт. Загальні положення”;
- ДСТУ 3974-2000 „Правила виконання дослідно-конструкторських робіт. Загальні положення”;
- ДСТУ 3918-1999 „Процеси життєвого циклу програмного забезпечення”;
- ДСТУ 3974-2000;
- ДСТУ 3396.0-96;

- ДСТУ 3396.1-96;
- ДСТУ 3396.2-97;
- НД ТЗІ 1.4-001-2000;
- НД ТЗІ 2.5-008-2002;
- НД ТЗІ 1.1-003-99;
- НД ТЗІ 2.5-004-99;
- НД ТЗІ 2.5-005-99;
- НД ТЗІ 3.7-001-99;
- НД ТЗІ 2.5-010-03;
- Порядку захисту інформації в інформаційно-телекомунікаційних системах (затверджено ДСТСЗІ СБУ 24.12.2001 р. №76);
- Положенню про державну експертизу в сфері технічного захисту інформації (затверджено ДСТСЗІ СБУ 29.12.199 р. № 62);
- Нормативно-правовим актами Міністерства охорони навколишнього природного середовища України;
- Конвенції ……..

5. Основні технічні вимоги до виконання роботи

5.1. Вимоги до АС, ПЗ ведення БД ……

НДР повинна виконуватись з урахуванням вимог ДСТУ 3973-2000, Постанови Кабінету Міністрів України від 04.02.1998 р. №121 „Про затвердження переліку обов'язкових етапів робіт під час проектування, впровадження та експлуатації систем і засобів автоматизованої обробки та передачі даних" та нормативно-правової бази, наведеної вище.

При створенні АС та ПЗ ведення БД …….. повинні бути забезпечені:

- єдина нормативно-довідкова інформація;
- можливість подальшого розширення;
- уніфікація інтерфейсів;
- принцип автоматизації ведення БД АС та ПЗ;
- персоніфікація інформації, яка надається користувачеві;
- збереження інформації при збоях та автоматичне її відновлення при зникненні причини збою;
- забезпечення захисту від НСД ресурсів АС та ПЗ.

5.2. Вимоги до структури та архітектури АС та ПЗ ведення БД ……

В результаті виконання робіт має бути створено технічне рішення, що забезпечує функціювання АС та ПЗ ведення БД ……, а також комплекс засобів захисту ресурсів від НСД на платформі створюваного за цією роботою ПЗ.

Архітектура системи має надати можливість створити та вести в автоматизованому режимі …….

При створенні системи будуть враховані вимоги інформаційної безпеки шляхом використання технологій протидії НСД до ресурсів АС та ПЗ.

5.3. Вимоги до складу та функций

До складу АС та ПЗ ведення БД ……повинні входити наступні функціональні підсистеми:

- формування ……на стадії обговорення;

  ……..

Функції користувачів АС та ПЗ повинні бути викладеними в експлуатаційній документації та Порядку застосування результатів НДР …….

5.4. _____ .

До складу АС та ПЗ ведення БД …… повинні увійти користувачі двох категорій: кінцеві користувачі, адміністратори АС та ПЗ та адміністратор безпеки.

Адміністратори АС та ПЗ та адміністратор безпеки повинні мати знання з адміністрування операційних систем.

До кінцевих користувачів АС та ПЗ ніяких додаткових вимог, крім знання загальних можливостей системи та правил техніки безпеки та експлуатації АС та ПЗ, не повинно висуватись.

5.5. _____ ***В имоги до надійності***

АС та ПЗ ведення БД ……. повинна забезпечити надійність зберігання інформації на рівні не нижче 99.8 % . Це повинно досягатися такими заходами:

- утворення та оновлення страхових копій на допоміжно-запам'ятовуючих пристроях;
- ведення протоколу поповнення БД;
- реалізацією контролю цілісності БД та дублювання найбільш цінної інформації;

- створення резервних копій на зовнішніх носіях, в т.ч. оптичних та магнітооптичних;
- телекомунікаційні засоби повинні забезпечити надійну передачу інформації по каналах зв'язку з можливістю автоматичного повторного з`єднання.

5.6. Вимоги з ергономіки та технічної эстетики

В АС та ПЗ ведення БД ……. повинен бути забезпечити дружній інтерфейс з користувачами.

5.7. _____

Вимоги до захисту інформації

СЗІ АС та ПЗ ведення БД …….. має відповідати наступним вимогам.

5.7.1. На рівні АС та ПЗ та компонентів повинна бути забезпечена можливість:

- ідентифікації користувачів;
- реєстрація та облік входу та виходу із системи користувачів;
- розмежування прав доступу на рівні документів;
- розмежування прав користувачів щодо модифікації атрибутів та полів реєстраційних та контрольних карток;
- розмежування прав доступу користувачів на рівні функціональних модулів та процедур;
- контролю цілісності даних;
- реєстрації спроб порушення режиму безпеки;
- резервного копіювання даних.

5.7.2. До складу АС та ПЗ має увійти КЗЗ, які відповідають вимогам до послуг безпеки:

- аутентифікація користувачів має здійснюватись на рівні СУБД та на прикладному рівні з використанням аутентифікації за ім`ям та паролем користувача СУБД;
- КЗЗ мають забезпечувати послуги цілісності та доступності з рівнем гарантії 2;
- КЗЗ повинні забезпечувати облік і реєстрацію подій, які пов'язані із безпосереднім доступом (спробами доступу) до інформації, здійснення періодичного контролю за такими подіями та забезпечувати захист

реєстраційної інформації від несанкціонованої модифікації, руйнування або знищення. Обсяг реєстраційної інформації повинен бути достатнім для встановлення причин та джерела виникнення зареєстрованої події;

- можливості виконання функцій та доступу до інформації БД повинні надаватись тільки зареєстрованим користувачам за умови їх достовірної аутентифікація;
- КЗЗ повинна виключати можливість порушення цілісності власного складу та складу окремих програмних компонентів, несанкціонованого використання ресурсів АС та ПЗ з метою отримання інформації або втручання в функціювання АС та ПЗ поза регламентом та правами доступу відповідних користувачів;
- права доступу користувачів повинні завірятися адміністратором АС та ПЗ з використанням механізмів та організаційних заходов.

5.7.3. Склад послуг безпеки, а також механізмів захисту, що реалізують кожну з послуг, визначається політикою безпеки інформації в ……...

5.7.4. КЗЗ повинні надавати можливість формування робочих груп з використанням засобів адміністрування відповідно до функцій, що необхідно виконувати конкретному користувачеві або групі користувачів.

5.8. Вимоги з інтеграції

5.8.1. АС та ПЗ ведення БД ……..формаційного обміну через інтерфейси та стандарти SQL та XML. Компоненти АС та ПЗ мають забезпечити безпосередню підтримку протоколів HTTP, HTTPS, SMTP.

5.8.2. АС та ПЗ ведення БД ……. повинні підтримувати глобальні зовнішні класифікатори, доступ до яких здійснюється через реляційні інтерфейси CLI/ODBC. Зміст зовнішніх класифікаторів має автоматично синхронізуватися з внутрішніми глобальними класифікаторами, які реплікується на всі рівні ієрархії системи.

5.8.3. ……..

5.8.4. АС та ПЗ ведення БД ……. має забезпечити можливість інтеграції з програмними та програмно-технічними засобами електронного підпису, які відповідають вимогам законодавчої та нормативної бази

України, для впровадження методів та технологій електронного підпису в електронному документообігу МОНПСУ.

5.9. Вимоги до патентної чистоти

В якості системного та прикладного ПЗ повинні використовуватися ліцензовані програмні продукти. При розробці АС та ПЗ ведення БД ……. контроль забезпечення патентної чистоти здійснюється на рівні наявності ліцензії на програмні продукти. Для розроблюваних компонентів ПЗ патентна чистота має забезпечуватися згідно встановленого порядку та чинного законодавства України.

5.10. Вимоги по техніці безпеки

Проектні рішення щодо створення АС та ПЗ ведення БД …… повинні передбачати заходи та процедури по експлуатації, які узгоджуються з вимогами “Правил охорони праці під час експлуатації електронно-обчислювальних машин”, затвердженим Наказом Комітету по нагляду за охороною праці України Міністерства праці та соціальної політики України від 10 лютого 1999 року №21.

5.11. Вимоги до навчання персоналу

Навчання адміністраторів та користувачів має провадиться на площах Замовника або Виконавця за узгодженням по таких програмах навчання: навчання адміністраторів АС та ПЗ ведення БД ……, навчання користувачів.

Програма навчання адміністраторів має включати підготовчий курс з адміністрування прикладних та системних програмних засобів, конфігурування системи тощо.

Програма навчання користувачів має включати підготовчий курс з основ використання АС та ПЗ та взаємодії з Інтернет-ресурсами.

Виконавець повинен провести навчання 2 адміністраторів та 2 користувачів за вказаними вище програмами.

5.12. Склад та зміст робіт по виконанню НДР

№	Назва етапу (підетапу)	Чим закінчується етап (звіт, макет,	Строки виконання

		конструкторська документація тощо)	
1	2	3	4
1.	Розроблення АС та ПЗ ведення БД ……	Звіт, погромне забезпечення, експлуатаційна та супровідна документація, акт здачі-приймання робіт.	2 місячі з початку виконання робіт (після підписання договору).
1.1.	Розробка проектної документації щодо створення ПЗ ведення БД …….	Техноробочий проект.	0,5 місячі з початку виконання робіт (після підписання договору).
1.2.	Розробка макету ПЗ ведення БД ……..	Макет програмного забезпечення. Проект експлуатаційної документації.	1 місяць з початку виконання робіт.
1.3.	Доопрацювання макету ПЗ ведення БД …….	Програмне забезпечення згідно з технічним завданням. Проект експлуатаційної документації.	0,5 місяці з завершення етапу 1.2 виконання робіт.
1.4.	Дослідна експлуатація ПЗ ведення БД ……..		0,5 місяці з завершення етапу 1.2 виконання робіт.
1.5.	Розроблення експлуатаційної та супровідної документації до програмного забезпечення щодо ведення БД ……...	Експлуатаційна та супровідна документація щодо ПЗ та інформаційної технології.	0,5 місяці з завершення етапу 1.2 виконання робіт.
1.6.	Створення СЗІ АС та ПЗ ведення БД ……...	СЗІ.	2 місячі з початку виконання

			робіт (після підписання договору).
1.7.	Введення до експлуатації програмного забезпечення щодо ведення БД …….	Методика та програма випробувань. Протокол випробувань. Акт введення в експлуатацію.	0,5 місяці з завершення етапу 1.5 виконання робіт.
1.8.	Навчання користувачів програмного забезпечення щодо ведення БД ……..	Програма навчання.	0,5 місяці з завершення етапу 1.7 виконання робіт.
1.9.	Впровадження ПЗ в …….		0,5 місяці з завершення етапу 1.7 виконання робіт.
1.10.	Авторське супроводження програмного забезпечення щодо ведення БД ……...		12 місяці з дня підписання акту здачі-приймання робіт за договором.

6. Вимоги щодо порядку приймання роботи

6.1. Порядок контролю та приймання НДР

6.1.1. Приймання АС та ПЗ ведення БД ……., відповідного ПЗ, відповідної СЗІ та НДР в цілому здійснюється комісією Замовника та затверджується актом здачі-приймання робіт.

Загальний порядок виконання і приймання АС та ПЗ повинен відповідати вимогам ГОСТ 34.601-90, ПЗ – з урахуванням вимог ДСТУ 2851-94 та ДСТУ 2853-94, СЗІ – вимог ДСТУ 3396.1-96, ДСТУ 3396.2-97 та відповідних НД ТЗІ СБ України;

Документування результатів випробувань повинно здійснюватися згідно з вимогами ГОСТ 34.603-92.

Створений комплекс засобів АС, ПЗ та СЗІ походить попередні випробування, які проводяться Виконавцем, та дослідну експлуатацію, які проводяться Виконавцем.

Випробування АС, ПЗ та СЗІ проводяться відповідно до Програм і методик, які розробляються Виконавцем та погоджуються Замовником.

6.1.2. Вимоги до складу та змісту робіт по підготовці АС та ПЗ ведення БД ……

Підготовка АС та ПЗ ведення БД …… до введення в дію полягає в виконанні наступних робіт:

- визначення та узгодження з Виконавцем дати введення в дію;
- тестування та перевірка Виконавцем разом з Замовником готовності технології передачі даних між визначеними підрозділами та користувачами;
- тестування та перевірка Виконавцем готовності всіх складових частин АС та ПЗ ведення БД до введення в експлуатацію;
- підготовка Виконавцем необхідної технічної та експлуатаційної документації;
- підготовка Виконавцем необхідних проектів розпорядчих документів.

При створенні АС та ПЗ ведення БД ……. повинні дотримуватися вимоги до обсягу й послідовності виконуваних робіт відповідно до Постанови Кабінету Міністрів України від .. ……. …. р. №121 “……..”.

Проектна документація повинна виконуватися відповідно до чинних в Україні нормативів і стандартів з урахуванням вимог вищезазначених нормативно-правових документів.

Документи щодо результатів НДР передаються Замовнику в двох примірниках на паперовому носії та в одному примірнику – на електронному носії.

Мова документів - українська.

7. Впровадження результатів роботи

Результати НДР використовуються у вигляді АС та ПЗ ведення БД …….

За результати роботи користувачі матимуть змогу взаємодіяти з інформаційним ресурсом АС та ПЗ ведення БД ……. з використанням технологій глобальної комп’ютерної мережі Інтернет.

Використання науково-технічної продукції здійснюється Замовником в складі…….

8. **Внесення змін та доповнень**

У разі зміни вимог цього ТЗ щодо архітектури та конфігурації систем, які можуть бути викликані непередбаченими поточними потребами Замовника, складається локальне ТЗ. Додаткові обсяги робіт, викликані цими обставинами, складають предмет окремих Договорів між Замовником та Виконавцем.

В ході виконання робіт розробляється часткове технічне завдання на створення комплексної системи захисту інформації АС та ПЗ ведення БД ……...

Всі уточнення та доповнення до цього Технічного завдання вносяться лише за узгодженням Замовника та Виконавця у письмовому вигляді.

Рекомендована література

1. .Соммервилл. Инженерия программного обеспечения. 6-е издание. - Издательский дом "Вильямс", 2002.

2. Ф.А.Новиков, Э.А.Опалева, Е.О.Степанов. Учебно- методическое пособие по дисциплине «Управление проектами и разработкой ПО» // [Електронний ресурс] - Режим доступу: http://window.edu.ru/resource/366/6G366/files/itmo3G5.pdf.

3. Технологии командной разработки программного обеспечения информационных систем // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/4806/1054/lecture/9998.

4. Д.Андреев. Организация процессов разработки программного обеспечения с использованием Team Foundation Server 2010 // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/649/505/info.

5. Інформаційна технологія. Системна та програмна інженерія. Процеси життєвого циклу програмних засобів ISO 12207.

6. Інформаційні технології. Системна та програмна інженерія. Вимоги та оцінка якості систем та програмного забезпечення. Моделі якості систем та програмних продуктів ISO 25010.

7. Інформаційні технології. Системна та програмна інженерія. Вимоги та оцінка якості систем та програмного забезпечення. Елементи показника якості ISO 25021.

8. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення ГОСТ 34.601-90.

9. Комплекс стандартів на автоматизовані системи. Технічне завдання створення автоматизованої системи ГОСТ 34.602-89.

10. Комплекс стандартів на автоматизовані системи. Види, комплектність та позначення документів під час створення автоматизованих систем [Електронний ресурс]: ГОСТ 34.201-89.

Лабораторна робота №11

«Дослідження методів оцінки розміру і повторного використання програмних засобів»

Мета роботи: засвоєння базових знань щодо дослідження методів оцінки розміру і повторного використання програмних засобів.

Підготовка до роботи: Вивчити і уяснити призначення і зміст,завдання, інструментальні засоби та основні процеси розробци документації на програмну продукцію (ПД).

Завдання:

14. Ознайомитись з методичною розробкою до лабораторноъ роботи.
15. Ознайомитись з рекомендованною літературою.
16. Ознайомитись з діючими стандартами.
17. Дослідити методи оцінки розміру і повторного використання програмних засобів.
18. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
19. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

10Назва та мета роботи.

2. Коротко теоретичні відомості

20. Завдання дослідження лабораторної роботи згідно до обраного варіанта.
21. Результати виконаного дослідження.
22. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

При оцінці вартості проекту і кількості часу, необхідного для його виконання, треба виконати два основних етапи:

1. Оцінювання розміру проекту.

2. Уявлення про ці розміри поряд з інформацією про інші фактори середовища дозволяє оцінити загальний об'єм трудовитрат і , відповідно, вартість робіт. Уточнення розмірів створюваного ПЗ передує цьому етапу кодування, виконуваного з метою реалізації вимог на практиці.

Одиниці вимірювання при оцінці розміру ПЗ

Яким чином можна визначити кількість рядків коду (*Lines of code, LOC*) до того, як вони фактично будуть написані або просто спроектовані?

Як показник кількість рядків коду може відображати величину трудовитрат, якщо не буде враховуватися складність виробленого продукту, здатності/стиль програміста або можливості мови програмування.

Взаємозв'язок між LOC і витраченими зусиллями не є лінійним. ***Середня продуктивність програміста складає близько 3000 рядків коду на одного програмісті на рік***.

Велике значення має набір функціональних властивостей і якість ПЗ, а не кількість рядків програмного коду.

Приклади одиниць вимірювання при оцінці розміру ПЗ:

- кількість рядків коду (*Lines Of Code, LOC*);
- функціональні точки;
- точки властивостей;
- кількість «бульбашок» на діаграмі потоку даних (*Data flow diagram, DFD*);
- кількість суттєвостей на діаграмі суттєвостей (*Entity relationship diagram, ERD*);
- кількість «квадратиків», що відповідають процесу/контролю (*PSPEC / CSPEC*) на структурному графіку;
- кількість різноманітних елементів у складі управлінської специфікації; - об'єм документації;
- кількість об'єктів, атрибутів і служб на об'єктній діаграмі.

Навички менеджменту IT-проектів

1. ***Створення структури пооперaційного переліку робіт*** – задача в рамках виконання проекту і виробництва продукту розбиваються на

достатньо малі складові елементи, розмір яких буде потім оцінюватися.

2. ***Документування планів*** – розбиті на складові елементи задачі в межах виконання проекту і виробництва продукту оцінюються з метою обрахунку величини трудовитрат і накладних витрат.

3. ***Оцінка вартості і трудовитрат, необхідних для завершення проекту*** – завдяки оцінці можливого розміру ПЗ можна думати про відносний розмір запланованих трудовитрат.

4. ***Складання графіку*** – на основі оцінки трудовитрат можна скласти графік.

5. ***Вибір метричних показників*** – на базі застосованих одиниць вимірювання формується метричний показник. Його достатньо обрати лише один раз, а потім посилатися.

Початковий етап визначення розмірів програмного продукту: оцінювання

Визначення розмірів програмного продукту являє собою і прогнозування структури продуктів, що поставляються, з метою задоволення вимог проекту. В процесі оцінювання також виконується прогнозування трудовитрат (ресурсів), необхідних для проектування продуктів, що поставляються. Після задач з прогнозування розмірів ПЗ слідують задачі по оцінці тривалості і вартості розробки, в процесі виконання яких виникає розподіл ресурсів, облік записів, а також складання робочого графіку.

Оцінка розміру ПЗ в процесі розробки проекту

WBS (Work Breakdown Structure або ІСР – ієрархічна структура робіт) проекту – це поділ проекту на конкретні результати, які мають бути досягнуті для досягнення цілей проекту.

Як тільки структура WBS буде розбита з урахуванням виділення самих нижніх рівнів, може бути створений «статистичний» показник розміру.

При цьому використовуються процеси вимірювання і підсумовування. Величина розміру кожного компоненту може бути отримана шляхом опитування експертів, які розроблюють подібні системи, або шляхом використання даних опитування потенціальних розробників подібних систем.

При підсумовування результатів вимірювання отриманий ітог буде називатися оцінкою розміру «знизу-угору». Покращена оцінка може бути отримана у тому випадку, якщо кожний оцінювач проведе оптимістичну, песимістичну і реалістичну оцінку розмірів. Потім формується бета-

розподіл шляхом помноження реалістичної оцінки розміру на 4, додавання оптимістичної і песимістичної оцінок з подальшим поділенням результату 6. Подібне зважене середнє значення є зручним в умовах природної невизначеності процесу оцінювання.

Наприклад, якщо даний віконний об'єкт відображається в структурі WBS для системи, яка підтримує код, що вимагається для реалізації процесу редагування в даному вікні, може займати від 200 до 400 рядків коду, причому, скоріше за все ця цифра стане близькою до 250. Враховуючи запропоновані оцінювачем песимістичні та оптимістичні сценарії, можна отримати наступні кінцеві оцінки:

ПО – песимістична оцінка розмірів;

ОО – оптимістична оцінка розмірів;

РО – реалістична оцінка розмірів;

*(ПО+ОО+(РО*4))/6= LOC*

*(200+400+(250*4))/6=266,67 =266 LOC*

Кількість тисяч рядків вихідного коду (*KSLOC*) є похідним від загальної метрики, введеної при оцінках продуктивності.

Зазвичай продуктивність визначається в *KSLOC/SM* або *KLOC/SM* (де *SM* -staff-month (людино-години).

Перший і другий стовбці таблиці рис. 1 представляють метод трансляції *SLOC, який застосовується і різноманітних мовах по відношенню до* середньої кількості рядків *SLOC* в базовій мові асемблеру.

Наприклад, приймемо, що операційна система, написана на мові С і яка нараховує 50 000 рядків *LOC*, буде перетворення в С++. Операційна система, яка містить 50 000 рядків *SLOC*, написаних на мові С, буде еквівалентною 125 000 рядкам у випадку використання мови асемблера (50000×2,5). Якщо 125 000 рядків операційної системи на мові асемблеру були переписані з урахуванням використання мови С++, отримаємо 125000/6, або 20833 *LOC* або 20,833 *SLOC*.

Рисунок 1 – Показники SLOC при перетворенні коду з мови на мову *Basic Assembler SLOC* з розрахунку на одну функціональну точку. При оцінці якості може застосовуватися співвідношення:

кількість дефектів/кількість рядків коду.

Велика величина бути свідчити про високу якість коду.

Язык программирования	Basic Assembler SLOC (уровень)	Средний показатель SLOC из расчета на одну функциональную точку
Basic Assembler	1	320
Autocoder	1	320
Macro Assembler	1,5	213
C	2,5	**128**-150
Пакетные файлы DOS	2,5	128
Basic	3	107
Макросы LOTUS	3	107
ALGOL	3	105-106
COBOL	3	105-107
FORTRAN	3	105-106
JOVIAL	3	105-107
Смешанные языки программирования (по умолчанию)	3	105
Pascal	3,5	91
COBOL (ANSI 85)	3,5	91
RPG	4	80
MODULA-2	4,5	80
PL/1	4,5	80
Параллельный Pascal	4	80
FORTRAN 95	4,5	71
BASIC (ANSI)	5	64
FORTH	5	64
LISP	5	64
PROLOG	5	64
LOGO	5,5	58
Расширенный общий LISP	5,75	56
RPG III	5,75	56
C++	**6**	**53**
JAVA	6	53
YACC	6	53
Ada 95	6,5	49
CICS	7	46
SIMULA	7	46
Языки баз данных	8	40
CLIPPER DB и dBase III	8	40
INFORMIX	8	40
ORACLEи SYBASE	8	40
Access	8,5	38
Dbase IV	9	36
FileMaker Pro	9	36
Языки поддержки принятия решений	9	35
FOXPRO 2.5	9,5	34
APL	10	32
Статистические языки (SAS)	10	32
DELPHI	11	29
Стандартные объектно- ориентированные языки	11	29
OBJECTTVE-C	12	27
Oracle Developer/2000	14	23
SMALLTALK	15	21

Рис.1 – Показники SLOC при перетворенні коду з мови на мову Basic Assembler SLOC з розрахунку на одну функціональну точку

Метод функціональних точок

При оцінці якості може застосовуватись співвідношення:

кількість дефектів/кількість рядків коду

Висока величина бути свідчити про високу якість коду.

Метод функціональних точок (*Function point, FP*) базується на тому, що розмір ПЗ краще за все оцінювати в термінах кількості і складності функцій, реалізованих у даному програмному коді, а не за допомогою кількості рядків коду. При використанні методу функціональних точок вимірюються категорії користувацьких бізнес-функцій.

Метод функціональних точок дозволяє виконати такі задачі:

- оцінити категорії користувацьких бізнес-функцій;
- вирішити проблему, пов'язану з намаганням застосувати одиниці вимірювання *LOC* на ранніх стадіях життєвого циклу розробки;
- визначати кількість і складність вхідних і вихідних даних, запити до бази даних, файли або структури даних, а також зовнішні інтерфейси, пов'язані з програмною системою.

Процес використання методу функціональних точок

На рис. 2 наведений робочий аркуш аналізу по методу функціональних точок.

Крок 1. Підрахунок функцій в кожній категорії.

Інструкція по визначенню рівня складності приведені на рис. 3. При підрахунку структур даних (файлів) необхідно враховувати таке:

- внутрішні файли являють собою логічні файли у складі програми;
- структури даних (раніше відомі як «файли») являють собою первину логічну групу користувацьких даних , які постійно знаходяться повністю у середині меж програмної системи;
- структури даних доступні для користувачів за допомогою вводу, виходу, опитувань або інтерфейсів.

Структури даних поділяються на прості, середні і складні.

Інструкції по визначенню ступеню складності приводяться у таблиці на рис. 4.

Шаг 1.	**Подсчет количества функций в каждой категории**			
Шаг 2.	**Применение весовых множителей сложности**			
	Простой	**Средний**	**Сложный**	**Функциональные точки**
Количество выводов	_*4	_*5	_*7	
Количество вводов	_*3	_*4	_*6	
Количество опросов выводов	_*4	_*5	_*7	
Количество опросов вводов	_*3	_*4	_*6	
Количество файлов	_*7	_*10	_*15	
Количество интерфейсов	_*5	_*7	_*10	
Общее количество функциональных точек				
Шаг 3.	**Применение факторов среды**			
Фактор среды				**Рейтинг (0,1,2,3,4,5)**
Каналы передачи данных				
Распределенные вычисления				
Требования к производительности				
Конфигурирование с ограничениями				
Частота транзакций				
Интерактивный запрос и/или запись				
Эффективность на уровне конечного пользователя				
Интерактивное обновление				
Сложная обработка				
Повторное использование				
Упрощение преобразования/установки				
Упрощение операции				
Используется на нескольких узлах				
Потенциал изменения функции				
Итого (N):				
Шаг 4.	**Вычисление корректировочного множителя сложности (CAF)**			
	CAF = 0,65 + (0,01*N) =			
Шаг 5.	**Вычисление скорректированных функциональных точек (AFF)**			
	AFP = FP (исходное число) * CAF =			
Шаг 6.	**Преобразование в строки LOC (дополнительно)**			
	LOC = AFP * LOC/AFP=			

Рис. 2. Робочий аркуш аналізу

Часть вывода	От 1 до 5 элементов	От 6 до 19 элементов	20 и более элементов данных
От 0 до 1	Простой 4	Простой 4	Средний 5
От 2 до 3	Простой 4	Средний 5	Сложный 7
4 и более	Средний 5	Сложный 7	Сложный 7
Часть ввода			
От 0 до 1	Простой 3	Простой 3	Средний 4
От 2 до 3	Простой 3	Средний 4	Сложный 6
4 и более	Средний 4	Сложный 6	Сложный 6

Рис. 3. Інструкція по визначенню рівня складності

Учитывайте логические взаимосвязи, а не физические типы записей	От 1 до 9 элементов данных	От 20 до 50 элементов данных	51 и более элементов данных
1 логическая запись формат / взаимосвязь	Простой 7	Простой 7	Средний 10
От 2 до 5 логических записей типа формат / взаимосвязь	Простой 7	Средний 10	Сложный 15
6 и более логических записей типа формат / взаимосвязь	Средний 10	Сложный 15	Сложный 15

Рис. 4. . Інструкція по визначенню рівня складності

При підрахунку кількості інтерфейсів необхідно враховувати таке:

- зовнішні фактори являють собою файли, сгенеровані комп'ютером, які використовуються програмою;
- інтерфейси являють собою дані (і систему управління), які зберігаються за межами програмної системи при виконанні оцінки;
- структури даних, які розділяються декількома системами, враховуються у вигляді інтерфейсів і структури даних;
- враховується кожний потік даних і управління у будь-якому напрямку в якості унікального інтерфейсу. Інтерфейси поділяються на прості, складні і середні.

Інструкції по визначенню рівня складності приведені у таблиці на рис. 5.

Учитывайте логические взаимосвязи, а не физические типы записей	От 1 до 9 элементов данных	От 20 до 50 элементов данных	51 и более элементов данных
1 логическая запись формат / взаимосвязь	Простой 5	Простой 5	Средний 7
От 2 до 5 логических записей типа формат / взаимосвязь	Простой 7	Средний 7	Сложный 10
6 и более логических записей типа формат / взаимосвязь	Средний 7	Сложный 15	Сложный 10

Рис.5 Вагові множники інтерфейсів в аналізі методом функціональних точок

Крок 2. Застосування вагових множників складності.

Необхідно помножити кожну величину певного типу (простий, середній, складний) в середині кожної категорії (вивід, введення, опитування, структура даних, інтерфейси) на відповідні вагові множники (рис. 3 - 5). в кожну категорію 8 додаються вихідні результати, виражені в кількості «фізичних функціональних точок».

Крок 3. Застосування факторів середовища.

Короткий опис виконання процесу зважування в середовищі. Використовуючи матеріал таблиці рис. 6, оцініть кожний фактор по шкалі від 0 до 5 (в даному випадку 0 значить неможливість застосування фактора).

Фактор среды	Рейтинг (0,1, 2,3,4, 5)
Каналы передачи данных	Данные или контрольная информация, принимаемые или получаемые через каналы передачи данных. Интерактивные системы всегда испытывают некоторое влияние со стороны каналов передачи данных
Распределенные вычисления	Приложение, использующее данные, которые хранятся, обрабатываются и к которым обеспечивается доступ в хранилище или системе обработки, отличающихся от используемых в основной системе
Требования к производительности	Требования, одобренные пользователем, которые были применены для получения высокой скорости передачи данных либо малого времени отклика
Конфигурирование с ограничениями	Приложение, выполняемое с применением интенсивно используемой, ограниченной или наполненной конфигурации
Частота транзакций	Высокий сетевой трафик, перегруженность экранов информацией и графикой, высокая частота обновления экрана
Интерактивный запрос и/или запись	Высокая степень интерактивности
Эффективность на уровне конечного пользователя	Необходимо дополнительно учитывать человеческий фактор
Интерактивное обновление	Динамическое обновление базы данных, распределенные базы данных
Сложная обработка	Высокая степень безопасности, обработка со множеством транзакций, сложные алгоритмы, логика, управляемая прерываниями
Повторное использование	Код, разработанный с целью повторного использования, должен обладать высоким качеством
Упрощение преобразования/установки	Процессы преобразования и установки требуют наличия документов по планированию, которые были протестированы
Упрощение операции	Эффективные, но простые процедуры запуска, резервирования, восстановления при появлении ошибок, а также завершения. Минимальное вмешательство со стороны пользователя
Используется на нескольких узлах	Учет различий в бизнес-функциях
Потенциал изменения функции	Модульность, управляемость с помощью таблиц, поддержка со стороны пользователей, возможности по формированию запросов и т.д

Рис. 6 - Опис факторів середовища

З метою отримання «ефекту присутності» на одному з країв спектру оцінювання, на рис. 7 приводяться приклади програмних систем з високим рейтингом – показники рейтингу від 4 до 5 на шкалі.

Просумуйте рейтинги факторів (*Fn*) з метою розрахунку кінцевого фактора впливу середовища (*N*).

N = sum (Fn)

Значення заповнюються в робочий аркуш аналізу по методу функціональних точок. Заповнений робочий аркуш наведений на рис. 8.

Крок 4. Розрахунок множника коригування складності.

Рівень невизначеності оцінок є функцією фази життєвого циклу. Це підтримує теорію емпіричних даних, відповідно до якої максимальний ступінь впливу факторів середовища на результати аналізу функціональних точок може складати +/- 35%. Причому розглядається максимальний ступінь впливу, так як аналіз функціональних точок

виконується на початку життєвого циклу. Як наслідок, у цьому випадку існує максимальна вірогідність неточності вимірювань. З метою компенсації подібної невизначеності у випадку недостатнього рівня знань множник корегування складності (*CAF*) буде застосовуватися к результативним значенням факторів середовища.

$CAF = 0.65 + (0.01 * N)$,

де N – сума зважених факторів середовища.

№	Фактор среды	Примеры высокопроизводительных систем
1	Сложные коммуникации данных	Программа, предназначенная для международного банка, которая выполняет денежные переводы из финансовых учреждений, разбросанных по всему миру
2	Распределенная обработка	Механизм поиска в Internet, при реализации которого обработка выполняется несколькими десятками компьютеров, работающими в параллельном режиме
3	Цели, требующие достижения максимальной производительности	Система контроля воздушного трафика, поддерживающая точное и своевременное определение позиции воздушного судна на основе показаний радиолокатора
4	Интенсивно используемое конфигурирование	Университетская система, реализующая одновременную регистрацию нескольких сотен студентов, находящихся в лаборатории
5	Высокая частота транзакций	Банковская программа, выполняющая миллионы транзакций за ночь с целью сведения баланса по всем счетам на начало нового рабочего дня
6	Интерактивный ввод данных	Программа утверждения закладных, с помощью которой клерки могут вводить данные в компьютерную систему в интерактивном режиме, используя бумажные анкеты, заполненные будущими домовладельцами
7	Проект, дружественный по отношению к пользователю	Программное обеспечение для компьютеризованных касс на станциях метро, оборудованных сенсорными экранами, позволяющими покупать билеты с помощью кредитных карточек
8	Интерактивное обновление данных	Авиационная система, позволяющая агентам туристических фирм заказывать авиарейсы и получать информацию о наличии свободных мест. Это ПО должно обеспечивать возможность блокировки и модификации некоторых записей в базе данных с целью предотвращения повторной продажи билетов на одни и те же места
9	Комплексная обработка	Медицинское ПО, которое на основе различных симптомов, выявленных у пациента, а также путем обширного логического вывода позволяет устанавливать предварительный диагноз
10	Повторное использование	Рабочий процессор, который может быть спроектирован таким образом, что его пакеты инструментов и меню могут встраиваться в другие приложения, такие как электронные таблицы или генератор отчетов
11	Облегченная установка	Приложение, выполняющее контроль оборудования, с помощью которого даже неспециалист может установить и настроить оборудование морской буровой вышки
12	Облегченное оперирование	Программа, предназначенная для анализа большого количества хронологических финансовых записей, которая должна оптимизировать обработку информации таким образом, чтобы не вынуждать операторов постоянно менять носители данных
13	Несколько узлов	ПО по поддержке платежных ведомостей для транснациональной корпорации, которое должно учитывать различные характеристики разнообразных стран, включая различную валюту и правила определения налоговых ставок
14	Гибкость	Программа финансового прогноза, которая может формировать месячные, квартальные либо годовые прогнозы, необходимые конкретному бизнес-менеджеру, который может нуждаться в

Рис. 7 – Фактори середовища

Так як ми маємо справу з 14 факторами середовища, кожний з який має вагу, що змінюється і діапазоні від 0 до 5, найменше значення для *N* може бути 0 (кожний з 14 факторів не має застосування); а найбільше значення для *N* може бути 70 (кожний з 14 факторів має максимальну вагу, дорівнює 5). Виходячи з цих граничних умов, приходимо до висновку, що мінімум *CAF* = 0.65 + (0,01 * 0) = 0,65. Максимум *CAF* = 0.65 + (0,01 * 70) = 1,35. (1.35 - 0.65 = 0.70). Рання оцінка розмірів і трудовитрат може відхилятись при умові використання фактору на величину +/-35%.

Крок 4 проілюстрований в таблиці на рис. 8.

Крок 5. Обрахування скорегованих функціональних точок.

Скореговані функціональні точки (AFP) – фізичні функціональні точки × *CAF*.

Крок 6. Перетворення у рядки LOC (додатково).

Метод функціональних точок забезпечує спосіб попередньої оцінки розміру потенціальних програм або програмних систем. При цьому здійснюється аналіз майбутніх функціональних властивостей з користувацької точки зору. Мови програмування є зовсім різним з точки зору їх характеристик, проте існує деяка середня кількість виконуваних операторів, необхідних для реалізації однієї функціональної точки.

Перетворення функціональних точок у рядки *LOC* може вимагатися в силу таких причин:

- з метою вимірювання і порівняння продуктивності або розміру програм або систем, які були написані на різних мовах програмування;
- з метою використання стандартних одиниць вимірювання для здійснення введення даних в інструментальні засоби оцінки;
- для перетворення розміру програми або додатку, написаних на будь-якій мові програмування, в еквівалентний розмір у випадку додатку, написаного на іншій мові програмування.

На рис. 8 ілюструється перетворення функціональних точок в *LOC* (перший і третій стовбці). В таблиці перераховані не всі мовні перетворення, схвалені *11-TUG*. Також необхідно відмітити, що цей перелік постійно поповнюється по мірі розробки нових мов програмування.

*LOC = скореговані функціональні точки * LOC на скореговану функціональну точку*

AFP x # LOC на AFP = LOC

Шаг 1.	Подсчет количества функций в каждой категории			
Шаг 2.	Применение весовых множителей сложности			
	Простой	Средний	Сложный	Функциональные точки
	1	2	3	(1+2+3)
Количество выводов	12* 4=48	11* 5=55	5* 7=35	48+55+35=138
Количество вводов	8* 3=24	9* 4=36	6* 6=36	96
Количество опросов выводов	5* 4=20	7* 5=35	3* 7=21	76
Количество опросов вводов	5* 3=15	8* 4=32	4* 6=24	71
Количество файлов	12* 7=84	3* 10=30	2* 15=30	144
Количество интерфейсов	9* 5=45	6* 7=42	4* 10=40	127
Общее количество функциональных «физических»точек				652
Шаг 3.	Применение факторов среды			
Фактор среды				Рейтинг (0,1,2,3,4,5)
Каналы передачи данных				5
Распределенные вычисления				5
Требования к производительности				3
Конфигурирование с ограничениями				0
Частота транзакций				5
Интерактивный запрос и/или запись				4
Эффективность на уровне конечного пользователя				5
Интерактивное обновление				4
Сложная обработка				2
Повторное использование				2
Упрощение преобразования/установки				3
Упрощение операции				4
Используется на нескольких узлах				5
Потенциал изменения функции				4
			Итого (N):	51
Шаг 4.	Вычисление корректировочного множителя сложности (CAF)			
	CAF = 0,65 +(0,01 *N) =0,65+(0,01*51)=1,16			
Шаг 5.	Вычисление скорректированных функциональных точек (AFP)			
	AFP = FP (исходное число) * CAF = 652 * 1,16 = 756,32			
Шаг 6.	Преобразование в строки LOC (дополнительно)			
	LOC = AFP * LOC/AFP= 756,32 * 128 = 96,808.96 LOC 128 из таблицы рис.16.2			

Рис. .8 – Заповнений робочий аркуш аналізу по методу функціональних точок

Порядок виконання лабораторної частини

1. Оснащення робочого місця

- методичні вказівки до виконання лабораторної роботи; - конспект лекцій з дисципліни;

- комп’ютер четвертого покоління і вище з операційною системою Windows.

2 Вимоги безпеки при проведенні заняття

При проведенні практичного заняття слід дотримуватися наступних вимог техніки безпеки:

- у комп’ютерному класі знаходитися лише у присутності викладача або лаборанта;

- не вмикати і не вимикати штекер з розетки самостійно;

- під час практичного заняття відкривати тільки вікна тих

комп’ютерних програм, які стосуються теми поточної роботи.

3 Виконання розрахунків

3.1 Контрольний приклад

1. *Дано:*

- песимістична оцінка розмірів = 500 LOC; - оптимістична оцінка розмірів = 3500 LOC; - реалістична оцінка розмірів = 2000 LOC. Визначити кількість рядків коду.

Розв’язання: Кількість рядків коду визначаємо за формулою:

*(ПО+ОО+(РО*4))/6*

*(500+3500+(2000*4))/6=2000 LOC*

2. *Дано:*

- програма на мові програмування Java;
- ця ж програма містить 500 LOC на мові С. Визначити кількість рядків програми на мові Java.

Розв’язання:

Визначаємо кількість рядків на асемблері: *500*2,5=1250 LOC*

Визначаємо кількість рядків *на Java 1250/6=208,33 LOC*

3. Визначити розмір ПЗ, якщо18 класів об’єктів та їх характеристики отримуються шляхом спостереження існуючих систем і реалізовані як середні величини в якості трьох процедур із розрахунку на один клас. Також шляхом спостереження за існуючими системами стало відомо, що середній розмір процедурної програми (мова С) дорівнює 40 *LOC*.

Розв’язання:

Застосуємо бліц-модель:

*Кількість процесів (класи обєкту) * кількість програм з розрахунку на один клас * розмір середньої програми = розрахунковий розмір*

Розмір ПЗ складатиме:

$18 * 3 * 40 = 2160 LOC.$

4. *Дано*:

- "бульбашок" = 5;
- кожна "бульбашка" процесу на рівні 0 DFD, приблизно відповідає 2 фактичним програмам на мові SQL;
- середній розмір програм в бібліотеці SQL дорівнює 100 рядкам LOC. Визначте розмір ПЗ

Розв'язання:

*Кількість процесів ("бульбашки" DFD) * кількість програм на "бульбашку" * розмір середньої програми = обчислювальний розмір*

Розмір ПЗ становитиме:

5 "більбашок" * 2 програми на "більбашку" * 100 рядків *LOC* на програму =1000 обчислених рядків *LOC*

5 *Дано*:

- в функціональній системі середня кількість модулів, необхідних для завершення 8 функцій = 3;
- розмір однієї функції = 100 *LOC*.

Визначте розмір ПЗ.

Розв'язання:

Розрахунок функціонально багатої системи виконаємо за формулою:

*WF * (кількість процесів і специфікацій контролю) * середня кількість LOC для даного модуля = LOC*

Розмір ПЗ становитиме:

3 модуля * 8 функцій * 100 LOC = 2,4 LOC

6 *Дано*:

- загальна кількість функціональних точок =100;
- сума зважених факторів середовища =20.

Визначте кількість рядків коду програми на мові *DELPHI* (*LOC*).

Розв'язання:

1) Розрахуємо корегуючого множника важкості:

CAF=0,65+(0,01 N)*

CAF=0,65+(0,01 20)=0,85*

2) Розрахуємо скореговані функціональні точки:

*AFP=FP(вихідне число)*CAF*

*AFP=100*0,85=85*

3) Перетворення у рядки LOC

LOC= AFP LOC/ AFP*

*LOC=85*29=2465 LOC*

7 Визначте кількість рядків коду програми на мові *C* (*LOC*), якщо в програмі будуть використовуватися прості алгоритми і прості дані, загальна кількість фізичних точок =100.

Розв'язання:

Скористаємося методом точок властивостей и заповнимо бланк робочого листа аналізу методом точок властивостей:

Фактори:	Середнє значення	Точки властивостей
Крок 1 Підрахунок кількості точок властивостей		
Всього: кількість фізичних точок властивостей		100
Крок 3 Складність зважування		
Крок 4 Оцінка факторів середовища корегування складності (CAF)		
Крок5 Обрахунок фактора		
Логічне значення (вибір одного з них):		
Прості алгоритми і обчислення	1	
Більшість простих алгоритмів	2	
Алгоритмі середньої складності	3	
Деякі складні алгоритми	4	
Багато складних алгоритмів	5	
Значення даних (вибір одного з них):		
Прості дані	1	
Числові змінні, але прості взаємозв'язки	2	
Декілька полів , файлів та інтерактивних зв'язків	3	
Складні файлові структури	4	
Дуже складні файлові структури і взаємозв'язки між даними	5	
Всього по CAF: CAF=1+1=2 – фактор корегування складності		
Крок 6 Множення фізичної кількості точок властивостей		
Фізичні FP* CAF=100*0,6=60 скореговані точки властивостей		
Крок 7 Перетворення у рядки коду (додатково)		
LOC(для мови С)= AFP* LOC/ AFP=60*128=7680 LOC		

3.2 Порядок виконання роботи

1 Визначити кількість рядків коду, якщо песимістична оцінка розмірів = 400 *LOC*, оптимістична оцінка розмірів = 2500 *LOC*, реалістична оцінка розмірів = 2000 *LOC*.

2 Визначити кількість рядків програми на мові Java, якщо відомо, що ця ж програма містить 400 *LOC* на мові *C*.

3 Визначити розмір ПЗ, якщо16 класів об'єктів та їх характеристики отримуються шляхом спостереження існуючих систем і реалізовані як середні величини в якості чотирьох процедур із розрахунку на один клас. Також шляхом спостереження за існуючими системами стало відомо, що середній розмір процедурної програми (мова С) дорівнює 40 *LOC*.

4 Визначте розмір ПЗ, якщо відомо, "бульбашок"=3 і кожна "бульбашка" процесу на рівні 0 *DFD*, приблизно відповідає 2 фактичним програмам на мові *SQL*, а також відомо, що середній розмір програм в бібліотеці SQL дорівнює 200 рядкам *LOC*.

5 Визначте розмір ПЗ, в функціональній системі середня кількість модулів, необхідних для завершення 6 функцій дорівнює 3, розмір однієї функції дорівнює 100 LOC.

6 Визначте кількість рядків коду програми на мові *DELPHI* (*LOC*), якщо дана загальна кількість функціональних точок =150, сума зважених факторів середовища =15.

7 Визначте кількість рядків коду програми на мові *Cobol* (*LOC*), якщо в програмі будуть використовуватися прості алгоритми і прості дані, загальна кількість фізичних точок =300.

Рекомендована література

1. Соммервилл. Инженерия программного обеспечения. 6-е издание. - Издательский дом "Вильямс", 2002.

2. Ф.А.Новиков, Э.А.Опалева, Е.О.Степанов. Учебно- методическое пособие по дисциплине «Управление проектами и разработкой ПО» // [Електронний ресурс] - Режим доступу: http://window.edu.ru/resource/366/6G366/files/itmo3G5.pdf.

3. Технологии командной разработки программного обеспечения информационных систем // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/4806/1054/lecture/9998.

4. Д.Андреев. Организация процессов разработки программного обеспечения с использованием Team Foundation Server 2010 // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/649/505/info.

5. Інформаційна технологія. Системна та програмна інженерія. Процеси життєвого циклу програмних засобів ISO 12207.

6. Інформаційні технології. Системна та програмна інженерія. Вимоги та оцінка якості систем та програмного забезпечення. Моделі якості систем та програмних продуктів ISO 25010.

7. Інформаційні технології. Системна та програмна інженерія. Вимоги та оцінка якості систем та програмного забезпечення. Елементи показника якості ISO 25021..

8. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення ГОСТ 34.601-90.

9. Комплекс стандартів на автоматизовані системи. Технічне завдання створення автоматизованої системи ГОСТ 34.602-89.

10. Комплекс стандартів на автоматизовані системи. Види, комплектність та позначення документів під час створення автоматизованих систем [Електронний ресурс]: ГОСТ 34.201-89.

Лабораторна робота №12

«Дослідження та оцінка тестового покриття»

Мета роботи: засвоєння базових знань щодо дослідження методів оцінки покриття вихідного коду тестами та проведення огляду інструментів автоматизації тестування.

Підготовка до роботи: Вивчити і уяснити призначення і зміст завдання до лабораторної роботи.

Завдання:

1. Ознайомитись з методичною розробкою до лабораторноъ роботи.
2. Ознайомитись з рекомендованною літературою.
3. Ознайомитись з діючими стандартами.
4. Дослідити методи оцінки покриття вихідного коду тестами та проведення огляду інструментів автоматизації тестування.
5. Виконати індивідуальне завдання до лабораторної роботи у відповідності до теми дипломного проекту (роботи)..
6. За результатами досліджень скласти звіт з обґрунтованими висновками.

Зміст звіту:

11Назва та мета роботи.

2. Коротко теоретичні відомості
7. Завдання дослідження лабораторної роботи згідно до обраного варіанта.
8. Результати виконаного дослідження.
9. Висновки по роботі.

Примітка

Оформлення звіту з лабораторного заняття необхідно виконувати відповідно до вимог ДСТУ 3008-95.

Теоретичні відомості

Тестове покриття - ступінь, визначений у відсотках, де деякі атрибути або їх комбінації були перевірені контрольними прикладами.

Виділяють такі види тестового *покриття:покриття вимог(* requirements coverage), *покриття* коду (code coverage), *покриття на базі аналізу потоку управління (control* flow coverage).

Покриття вимог - оцінка покриття тестами функціональних та нефункціональних вимог до програмного продукту. Для вимірювання цього показника необхідно: проаналізувати безліч вимог до продукту та розбити їх на окремі пункти; пов'язати кожен пункт із тестовими випадками, що перевіряють його. Зв'язки між пунктами вимог та конкретними тестовими випадками утворюють *матрицю трасування.* У випадку покриття вимог $T_{r\,\mathrm{cov}}$ обчислюється як

$$T_{r\,\mathrm{cov}} = \left(L_{\mathrm{cov}} / L_{\mathrm{total}}\right) \bullet 100\%. \qquad (1)$$

L_{cov} - кількість вимог, що перевіряються тестовими випадками; L_{total} - Загальна кількість вимог.

Оскільки розглянутий вид тестового покриття не враховує кінцеву реалізацію алгоритмів, покриття вимог може залишити неохопленими деякі фрагменти вихідного коду.

Тестовий випадок (test case) - структура, що описує сукупність дій, конкретних умов та параметрів, необхідних для перевірки функціональної вимоги або її частини. Тестовий випадок має уніфікований опис, що включає список дій, очікуваний від них результат та результат тесту. Методики розробки тестових випадків на основі наявних вимог розглядаються у тест-дизайні.

Тест дизайн - початковий етап тестування програмної продукції, що включає планування та проектування тестових випадків.

З методикою практичного застосування тест дизайну та прикладами тестових випадків можна ознайомитись за посиланням http://www.protesting.ru/testing/testdesign_practice.html.

Покриття коду $T_{c\,\mathrm{cov}}$ - другий вид тестового покриття - є найпростішою метрикою та визначається виходячи з формули

$$T_{c\,\mathrm{cov}} = \left(L_{tc} / L_{code}\right) \bullet 100\% \qquad (2)$$

L_{tc} - число виконаних рядків вихідного коду; L_{code} - загальне число рядків.

В якості прикладу на рис. 1 наведено фрагмент коду, що повертає період у розвитку людини на основі статі та віку. Для обчислення покриття коду взято два тести:

1) вихідні дані: "*female*", 17; результат містить "*girl*";
2) вихідні дані: "*male*", 16; результат містить "*teenager*".

При наявних вхідних даних $T_{\text{code}} = (6/20) \cdot 100\% = 30\%$.

Рис. 1. Приклад 30% покриття коду

Більш детальну інформацію про тестове покриття можна отримати на основі *тестування потоків управління (control flow testing*).

Тестування потоків управління - техніка структурного тестування (тестування методом «білої скриньки»), суть якої у визначенні шляхів виконання вихідного коду та проектуванні тестових випадків для їх [шляхів] покриття. Цей вид тестування використовує *граф потоку управління*- множина всіх можливих шляхів виконання програми, представлена у вигляді орієнтованого графа.

Для тестування потоків керування вводиться сім рівнів тестового покриття (табл. 1).

Величина покриття на будь-якому рівні визначається як відношення перевірених контрольними прикладами атрибутів (альтернатив, умов, умов альтернатив, множинних умов або шляхів) до їх загального числа, тобто. аналогічно формулам (1) та (2).

На псевдоприкладах розглянемо ключові рівні покриття.

Таблиця 1. Рівні тестового покриття

Рівень	Найменування	Примітка
0	"Тест whatever you test, users will test the rest"	"Тестуйте, що ви хочете, решту протестують користувачі"
1	Покриття операторів	Кожен оператор виконаний як мінімум один раз
2	Покриття альтернатив (гілок)	Перевіряється, якою мірою покриті альтернативні проходи в програмі. Тобто. кожен вузол з розгалуженням (альтернатива) виконаний як мінімум один раз (хоча б раз перевіряється кожна точка розгалуження алгоритму)
3	Покриття умов	Кожен варіант умови (*true ma false*) має бути виконаний мінімум один раз
4	Покриття умов альтернатив	Сумарне застосування рівнів з другого до третього
5	Покриття множинних умов	Сумарне застосування рівнів з другого до четвертого. Тобто. оцінюється покриття всіх можливих комбінацій умов
6	Покриття нескінченного числа шляхів	Задається максимальна глибина оцінки покриття за наявності довгого циклу або рекурсії
7	Покриття шляхів	Перевірці підлягають усі шляхи без винятку Найвищий рівень покриття, використовуючи який можна сказати, що програма на 100% покрита тестами

Приклад 1

```
if a then
  x++
else
  x--
endif
```

Варіанти вхідних даних

a = true

a = false

Для покриття операторів (рівень 1) x++ і x-- у прикладі 1 потрібно перевірити всього два варіанти вхідних даних.

Приклад 2 >
Покриття умов
a = true, b = true, c = true
a = false, b = false, c = false

if (a or b) and c then
Покриття умов альтернатив
a= false, b = false, c=true
a = true, b = false, c = true
a = false, b = true, c = true
a = false, b = true, c = false

У прикладі 2 для покриття умов (прохід гілкою if та інший варіант) достатньо взяти два набори тестових даних, значення b не важливо за умови a = true. Для покриття умов альтернатив потрібно більше варіантів вхідних даних, оскільки важливо, щоб кожну умову всередині альтернативи було виконано.

Покриття багатьох умов вважається 100%, якщо вхідні дані охопили всі можливі комбінації умов. 100% покриття множинних умов для прикладу 3 гарантує перевірка восьми варіантів (2з) вхідних даних.

Приклад 3 >
a = false, b = false, c = false
a = false, b = false, c = true
a = false, b = true, c = false
a = false, b = true, c = true

if (a or b) i c then
a = true, b = false, c = false
a = true, b = false, c = true
a = true, b = true, c = false
a = true, b = true, c = true

З метою автоматизації тестування інструменти аналізу покриття коду вбудовані в багато *IDE* (*Integrated Development Environment*, інтегроване середовище розробки), наприклад *Visual Studio.* Існують і приватні рішення - утиліти *(gcov,tcov,lcov,codecov)* та системи аналізу покриття програмного коду *(Testwell CTC++),* написаного різними мовами.

Порядок виконання лабораторної роботи

Завдання щодо визначення тестового покриття виконуються на прикладі логічно завершеного фрагмента програмного коду (фрагмент лабораторної або

курсової роботи) розміром 60-100 рядків. Фрагмент тексту програми обов'язково має містити умовні оператори.

1. Побудуйте набори вхідних даних, які разом зне- друкують покриття коду не менше 30%. У звіті перерахуйте ці набори, а також рядки коду, які вони покривають. Обчисліть точне значення покриття коду альтернатив.
2. Визначте набори вхідних даних, які разом зне- друкують покриття альтернатив не менше 30%. У звіті перерахуйте ці набори, вкажіть, які вони покривають. Обчисліть точне значення покриття альтернатив.
3. Пункт 2 зробіть для третього та четвертого рівнів тестового покриття.
4. Перевірте інструменти автоматизації тестування. У звіт зробіть приклади таких інструментів та їх короткий опис.

Факультативно. Використовуйте інструменти для автоматичного обчислення тестового покриття на практиці. Наведіть короткий опис своїх дій. Порівняйте з результатами п. 1 та 2.

Рекомендована література

1. Соммервилл. Инженерия программного обеспечения. 6-е издание. - Издательский дом "Вильямс", 2002.
2. Ф.А.Новиков, Э.А.Опалева, Е.О.Степанов. Учебно- методическое пособие по дисциплине «Управление проектами и разработкой ПО» // [Електронний ресурс] - Режим доступу: http://window.edu.ru/resource/366/6G366/files/itmo3G5.pdf.
3. Технологии командной разработки программного обеспечения информационных систем // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/4806/1054/lecture/9998.
4. Д.Андреев. Организация процессов разработки программного обеспечения с использованием Team Foundation Server 2010 // [Електронний ресурс] - Режим доступу: http://www.intuit.ru/studies/courses/649/505/info.
5. Інформаційна технологія. Системна та програмна інженерія. Процеси життєвого циклу програмних засобів ISO 12207.
6. Інформаційні технології. Системна та програмна інженерія. Вимоги та оцінка якості систем та програмного забезпечення. Моделі якості систем та програмних продуктів ISO 25010.

7. Інформаційні технології. Системна та програмна інженерія. Вимоги та оцінка якості систем та програмного забезпечення. Елементи показника якості ISO 25021..

8. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення ГОСТ 34.601-90.

9. Комплекс стандартів на автоматизовані системи. Технічне завдання створення автоматизованої системи ГОСТ 34.602-89.

10. Комплекс стандартів на автоматизовані системи. Види, комплектність та позначення документів під час створення автоматизованих систем [Електронний ресурс]: ГОСТ 34.201-89.

Правила оформлення і зарахування результатів виконання лабораторної роботи

Правила оформлення і надання звіту про виконання лабораторної роботи:

1. Поля: зверху 20, знизу - 20, зліва - 30, справа - 15 мм.
2. Шрифт - Times New Roman розмір -14.
3. Міжрядковий інтервал - 1.15.
4. Відступ 10 смм.
5. Нумерацію сторінок виконувати знизу справа. На першому аркуші номер не проставляти.
6. Заголовки розділів виконувати напівжирним стилем великими літерами.
7. Заголовки підрозділів виконувати напівжирним стилем з курсивом малими літерами.
6. Звіт викладається вигляді файлу в форматі .doc або docx на електронний ресурс Elearn.

До захисту лабораторних робіт не допускаються студенти, звіти яких вміщують такі недоліки:

1. Оформлення виконано не у відповідності до Правил оформлення і надання звіту.
2. Вирішені в звіті завдання не відповідають варіанту завдання студента.
3. Фрагмент звіту (починаючи від десяти йдучих підряд слів) або звіт в цілому одного студента є копією фрагменту звіту або усього звіту іншого студента.
4. Звіт викладено на електронний ресурс Elearn після закінчення семестру.

Лабораторні роботи оцінюються і зараховуються тільки після процедури захисту.

Приклад тестового завдання

1. Альфа-тестування (*alpha testing*) **- це:**
а) тестування при прийманні або димове тестування (smoke testing);
б)тестування нової функціональності (new feature testing);
в) регресивне тестування (regression testing);
г) тестування при здачі (*acceptance testing*);
д) а) і в);
є) б) і в);
ж) б - г);
з) а) - г);
і) а) і б);
к) немає правильної відповіді.

2. Тестуванні швидкодії включає напрямки:
а) навантажувальне тестування;
б) стрес-тестування;
в) тестування стабільності;
г) конфігураційне тестування;
д) а) і в);
є) б) і в);
ж) б - г);
з) а) - г);
і) а) і б);
к) немає правильної відповіді.

3. *Верифікація (Verification)* — це:
а) процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази;
б) визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи;
в) документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення;
г) етап процесу тестування програмного забезпечення, на якому

проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування;
д) документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини;
є) документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result);
ж) одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується;
з) рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття;
і) час від початку проходження кроків тест кейса до отримання результату тесту;
к) а) і г);
л) б), д), ж) і з);
н) б), в), і) і д) - з);
о) а) - є) і і);
п) в) - ж);
р) а) - д);
с) немає правильної відповіді.

4. Ефективність програмних систем (*efficiency*) **– це:**
а) властивості ПС, які забезпечують користувачу можливості зрозуміти, чи дійсно вона може задовільнити його потреби, як вона може бути використана для розв’язання визначених задач і які умови її використання;
б) набір атрибутів, які відносяться до об’єму робіт, які необхідні для виконання та індивідуальної оцінки такого виконання визначеним або передбачуваним колом користувачів;
в) набір атрибутів, які відносяться до здатності ПЗ бути перенесеним з одного оточення в інше;
г) набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об’ємом використаних ресурсів при встановлених умовах;
д) набір атрибутів, які відносяться до об’єму робіт, які необхідні для проведення конкретних змін (модифікацій);
є) а) і в);

ж) б) і в);
з) б), в), г), д);
і) а) - г);
к) а) і б);
л) а) - д);
м) немає правильної відповіді.

5. Тестування програмних систем (*software testing*) **– це:**
а) процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись;
б) процес пошуку помилок або інших дефектів, так і випробування програмних складових із метою оцінки;
в) процес перевірки відповідності заявлених до продукту вимог і реально реалізованої функціональності, здійснюваний шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином;
г) а) і в);
д) б) і в);
є) а) - в);
ж) немає правильної відповіді.

6. Процес супроводження програмних засобів у відповідноті до стандарту ISO 14764 – це:
а) коригування, вдосконалення продукту для усунення виявлених помилок або нереалізованих задач;
б) адаптація, підлаштування продукту до умов експлуатації, що змінилися, або в новому середовищі виконання;
в) поліпшення, еволюційна зміна продукту для підвищення продуктивності або рівня супроводу;
г) перевірка ПЗ, пошук і виправлення помилок при експлуатації системи;
д) а) і в);
є) б) і в);
ж) б - г);
з) а) - г);
і) а) і б);
к) немає правильної відповіді.

7. Рефакторінг – це;
а) удосконалення застарілого ПЗ шляхом його реорганізації або

реструктуризації, а також перепрограмування окремих елементів або настроювання параметрів на іншу платформу, середовище виконання зі збереженням зручності його супроводу;
б) відновленні специфікації (графів викликів, потоків даних і ін.) за отриманим кодом системи для її аналізу на більш високому рівні. Відновлюється ідентифікація компонентів і зв'язків між ними для забезпечення перепрограмування системи на нову платформу;
в) реорганізація програмного коду для поліпшення характеристик і показників якості об’єктно-орієнтованих і компонентних програм без зміни їх поведінки;
г) б) і в);
д) а) - в);
є) а) і б);
ж) а) і в);
з) немає правильної відповіді.

8. Переносимість (мобільність) систем (*portability*) **– це:**
а) властивості ПС, які забезпечують користувачу можливості зрозуміти, чи дійсно вона може задовільнити його потреби, як вона може бути використана для розв’язання визначених задач і які умови її використання;
б) набір атрибутів, які відносяться до об’єму робіт, які необхідні для виконання та індивідуальної оцінки такого виконання визначеним або передбачуваним колом користувачів;
в) набір атрибутів, які відносяться до здатності ПЗ бути перенесеним з одного оточення в інше;
г) набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об’ємом використаних ресурсів при встановлених умовах;
д) набір атрибутів, які відносяться до об’єму робіт, які необхідні для проведення конкретних змін (модифікацій);
є) а) і в);
ж) б) і в);
з) б), в), г), д);
і) а) - г);
к) а) і б);
л) а) - д);
м) немає правильної відповіді.

9. Зручність застосування програмних систем (Практичність) (*usability*) **– це:**

а) властивості ПС, які забезпечують користувачу можливості зрозуміти, чи дійсно вона може задовільнити його потреби, як вона може бути використана для розв'язання визначених задач і які умови її використання;

б) набір атрибутів, які відносяться до об'єму робіт, які необхідні для виконання та індивідуальної оцінки такого виконання визначеним або передбачуваним колом користувачів;

в) набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об'ємом використаних ресурсів при встановлених умовах;

г) набір атрибутів, які відносяться до співвідношення між рівнем якості функціонування ПЗ і об'ємом використаних ресурсів при встановлених умовах;

д) набір атрибутів, які відносяться до об'єму робіт, які необхідні для проведення конкретних змін (модифікацій);

є) а) і в);

ж) б) і в);

з) б), в), г), д);

і) а) - г);

к) а) і б);

л) а) - д);

м) немає правильної відповіді.

10. Час Проходження Тест Кейса (*Test Case Pass Time*) **— це:**

а) процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази;

б) визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи;

в) документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення;

г) етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до

визначених раніше критеріями якості та цілями тестування;
д) документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини;
є) документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result);
ж) одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується;
з) рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття;
і) час від початку проходження кроків тест кейса до отримання результату тесту;
к) а) і г);
л) б), д), ж) і з);
н) б), в), і) і д) - з);
о) а) - є) і і);
п) в) - ж);
р) а) - д);
с) немає правильної відповіді.

11. Рівні інтеграційного тестування:
а) компонентний інтеграційний рівень (Component Integration testing) - перевіряється взаємодія між компонентами системи після проведення компонентного тестування;
б) системний інтеграційний рівень (System Integration Testing) - перевіряється взаємодія між різними системами після проведення системного тестування;
в) знизу вгору (*Bottom Up Integration*);
г) зверху вниз (*Top Down Integration*);
д) великий вибух (*«Big Bang» Integration*);
к) а) і г);
л) б), д), ж) і з);
м) б), в) і д) - з);
н) а), б);
о) в) - ж);
п) а) - д);
р) немає правильної відповіді.

12. План Тестування (*Test Plan*) **— це:**

а) процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази;

б) визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи;

в) документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення;

г) етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування;

д) документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини;

є) документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result);

ж) одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується;

з) рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття;

і) час від початку проходження кроків тест кейса до отримання результату тесту;

к) а) і г);

л) б), д), ж) і з);

н) б), в), і) і д) - з);

о) а) - є) і і);

п) в) - ж);

р) а) - д);

с) немає правильної відповіді.

13. ***Тестовий випадок* (*Тест кейс/Test Case*) **— це:**

а) процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим

на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази;
б) визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи;
в) документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення;
г) етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування;
д) документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини;
є) документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result);
ж) одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується;
з) рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття;
і) час від початку проходження кроків тест кейса до отримання результату тесту;
к) а) і г);
л) б), д), ж) і з);
н) б), в), і) і д) - з);
о) а) - є) і і);
п) в) - ж);
р) а) - д);
с) немає правильної відповіді.

14. *Тестування інтерфейсу користувача* (*UI testing*) **–це:**.
а) тестування безпеки (security testing);
б) тестування локалізації (localization testing);
в) тестування сумісності (compatibility testing);
г) а) - в);
д) б) і в);

є) а) і б); ж) а) і в); з) немає правильної відповіді.
15. В прооцесі тестування може оцінюватись: а) відповідність вимогам, якими керувалися проектувальники та розробники; б) правильна відповідь для усіх можливих вхідних даних; в) виконання функцій за прийнятний час; г) практичність; д) сумісність із програмним забезпеченням та операційними системами; є) відповідність задачам замовника; є) а) і в); ж) б) і в); з) б), в), г), д); і) а) - є); к) в) - є); л) а) - д); м) немає правильної відповіді.
16. Деталізація Тест Кейсів (*Test Case Specification*) **— це:** а) процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази; б) визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи; в) документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків із варіантами їх вирішення; г) етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування; д) документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини; є) документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result);

ж) одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується;
з) рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття;
і) час від початку проходження кроків тест кейса до отримання результату тесту;
к) а) і г);
л) б), д), ж) і з);
н) б), в) і д) - з);
о) а) - є);
п) в) - ж);
р) а) - д);
с) немає правильної відповіді.

17. Регресивне тестування включає:
а) *new bug-fix* — перевірка виправлення знайдених дефектів;
б) *old bug-fix* — перевірка, що виявлені раніше й виправлені дефекти не відтворюються в системі знову;
в) *side-effect* — перевірка того, що не порушилася працездатність працюючої раніше функціональності, якщо її код міг бути зачеплений під час виправлення деяких дефектів в іншій функціональності;
г) б) і в);
д) а) - в);
є) а) і б);
ж) а) і в);
з) немає правильної відповіді.

18. Тест дизайн (*Test Design*) **— це:**
а) процес оцінки системи або її компонентів із метою визначити чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу. Тобто чи виконуються цілі, терміни, завдання з розробки проекту, визначені на початку поточної фази;
б) визначення відповідності розроблюваного програмного забезпечення очікуванням і потребам користувача, вимогам до системи;
в) документ, що описує весь обсяг робіт із тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки

ризиків із варіантами їх вирішення;
г) етап процесу тестування програмного забезпечення, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування;
д) документ, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації тестованої функції або її частини;
є) документ, що описує ситуацію або послідовність дій (Steps), що призвела до некоректної роботи об'єкта тестування (Misbehavior), із зазначенням причин та очікуваного результату (Expected Result);
ж) одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується;
з) рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття;
і) час від початку проходження кроків тест кейса до отримання результату тесту;
к) а) і г);
л) б), д), ж) і з);
н) б), в), і) і д) - з);
о) а) - є) і і);
п) в) - ж);
р) а) - д);
с) немає правильної відповіді.

19. Реверсна інженерія – це;
а) удосконалення застарілого ПЗ шляхом його реорганізації або реструктуризації, а також перепрограмування окремих елементів або настроювання параметрів на іншу платформу, середовище виконання зі збереженням зручності його супроводу;
б) відновленні специфікації (графів викликів, потоків даних і ін.) за отриманим кодом системи для її аналізу на більш високому рівні. Відновлюється ідентифікація компонентів і зв'язків між ними для забезпечення перепрограмування системи на нову платформу;
в) реорганізація програмного коду для поліпшення характеристик і показників якості об’єктно-орієнтованих і компонентних програм без зміни їх поведінки;
г) б) і в);

д) а) - в); є) а) і б); ж) а) і в); з) немає правильної відповіді.
20. Реінженерія – це; а) удосконалення застарілого ПЗ шляхом його реорганізації або реструктуризації, а також перепрограмування окремих елементів або настроювання параметрів на іншу платформу, середовище виконання зі збереженням зручності його супроводу; б) відновленні специфікації (графів викликів, потоків даних і ін.) за отриманим кодом системи для її аналізу на більш високому рівні. Відновлюється ідентифікація компонентів і зв'язків між ними для забезпечення перепрограмування системи на нову платформу; в) реорганізація програмного коду для поліпшення характеристик і показників якості об’єктно-орієнтованих і компонентних програм без зміни їх поведінки; г) б) і в); д) а) - в); є) а) і б); ж) а) і в); з) немає правильної відповіді.