МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни

“Професійна практика програмної інженерії”

для студентів спеціальності
121 “Інженерія програмного забезпечення”
(всіх форм навчання)

2016

Методичні вказівки до виконання лабораторних робіт з дисципліни “Професійна практика програмної інженерії” для студентів спеціальності 121 “Інженерія програмного забезпечення” (всіх форм навчання) / О.О. Олійник, А.О. Олійник, В.М. Льовкін. – Запоріжжя : ЗНТУ, 2016. – 51 с.

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

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

Відповідальний за випуск: С.О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “16” серпня 2016 р.

ЗМІСТ

ВСТУП

Дане видання призначене для вивчення та практичного засвоєння студентами всіх форм навчання основ професійної практики програмної інженерії.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. Дані методичні вказівки містять тільки основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт, тому для виконання лабораторної роботи та при підготовці до її захисту необхідно ознайомитись з конспектом лекцій та опрацювати весь необхідний матеріал, наведений в переліку рекомендованої літератури, використовуючи також статті в інтернет-виданнях та актуальних наукових журналах з програмної інженерії.

Для одержання заліку з кожної роботи студент повинен у відповідності зі всіма наведеними вимогами розробити програмне забезпечення та оформити звіт, після чого продемонструвати на комп'ютері розроблене програмне забезпечення з виконанням всіх запропонованих викладачем тестів.

Звіт виконують на білому папері формату А4 (210 х 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші вміщують у канцелярський файл.

Усі завдання повинні виконуватись студентами індивідуально і не містити ознак плагіату як в оформленому звіті так і в розробленому програмному забезпеченні.

Під час співбесіди при захисті лабораторної роботи студент повинен виявити знання щодо мети роботи, теоретичного матеріалу, методів виконання кожного етапу роботи, змісту основних розділів звіту з демонстрацією результатів на конкретних прикладах, практичних прийомів використання теоретичного матеріалу в розробленому програмному забезпеченні. Студент повинен вміти обґрунтувати всі прийняті ним рішення та правильно аналізувати і використовувати на практиці отримані результати.

Для базової самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

1 ЛАБОРАТОРНА РОБОТА № 1 РОЗРОБЛЕННЯ ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Мета роботи

Засвоїти основні принципи та засади оформлення технічного завдання на розробку програмного забезпечення.

1.2 Короткі теоретичні відомості

1.2.1 Призначення технічного завдання

Технічне завдання (ТЗ) є основним документом усього проекту та усіх взаємовідносин замовника і розробника. Коректне ТЗ, написане й погоджене усіма зацікавленими та відповідальними особами, є запорукою успішної реалізації проекту.

Великі проекти вимагають серйозного проектного дослідження. Як правило, на ці дослідження виділяється окремий бюджет і часом не менший, ніж на безпосередньо розробку проекту. Часто доводиться проектні дослідження зводити до мінімуму або частину досліджень проводити безкоштовно в надії на одержання великого проекту, що в остаточному підсумку може негативно позначитися на ефективності робіт.

Як правило, замовник не є професіоналом в області високих технологій, і завдання ним ставиться в загальному вигляді (наприклад: «Ми б прагли побачити от це, може це, а може ще й це.»). У такому випадку виконавець може сам запропонувати варіанти, черговість і етапність розв'язання поставленого завдання.

Після того як обговорено основні завдання й загалом стає зрозуміло, чого прагне замовник, необхідно скласти й погодити технічне завдання.

1.2.2 Структура технічного завдання

Згідно з ГОСТом 19.201–78 ТЗ повинно містити такі розділи:
– вступ;
– підстави для розробки;
– призначення розробки;
– вимоги до програми чи програмному виробу;

– вимоги до програмної документації;
– техніко-економічні показники;
– стадії та етапи розробки;
– порядок контролю та приймання.

У ТЗ допускається включати додатки.

У залежності від особливостей програми чи програмного виробу допускається уточнювати зміст розділів, вводити нові розділи чи поєднувати окремі з них.

У розділі «Вступ» вказують найменування, коротку характеристику області застосування програми чи програмного виробу та об'єкта, у якому використовують програму чи програмний виріб.

У розділі «Підстави для розробки» повинно бути зазначено:

– документ (документи), на підставі якого ведеться розроблення;

– організація, що затвердила цей документ, і дата його затвердження;

– найменування і (або) умовне позначення теми розробки.

У розділі «Призначення розробки» повинно бути зазначене функціональне та експлуатаційне призначення програми чи програмного виробу.

Розділ «Вимоги до програми чи програмного виробу» повинен містити наступні підрозділи:

– вимоги до функціональних характеристик;
– вимоги до надійності;
– умови експлуатації;
– вимоги до складу та параметрів технічних засобів;
– вимоги до інформаційної та програмної сумісності;
– вимоги до маркування та упакування;
– вимоги до транспортування та збереження;
– спеціальні вимоги.

У підрозділі «Вимоги до функціональних характеристик» повинно бути зазначено вимоги до складу виконуваних функцій, організації вхідних і вихідних даних (перелічені всі вхідні й вихідні дані та зазначено всі відомі вимоги до їх організації), тимчасових характеристик тощо. У даному підрозділі обов'язково має бути зазначено, якими групами користувачів та яким чином (з зазначенням відповідних функцій програми) планується використання програмного забезпечення.

У підрозділі «Вимоги до надійності» повинно бути зазначено вимоги до забезпечення надійного функціонування (забезпечення стійкого функціонування, контролю початкової та вихідної інформації, часу відновлення після відмовлення тощо). У даному підрозділі визначається, які саме збійні ситуації, ситуації невірного введення даних користувачем тощо мають оброблятися програмою.

У підрозділі «Умови експлуатації» повинно бути зазначено умови експлуатації (температура навколишнього повітря, відносна вологість тощо для обраних типів носіїв даних), при яких повинні забезпечуватися задані характеристики, вид обслуговування, необхідна кількість і кваліфікація персоналу (з зазначенням задач, які такий персонал повинен виконувати).

У підрозділі «Вимоги до складу і параметрів технічних засобів» указують необхідний склад технічних засобів із зазначенням їхніх основних технічних характеристик.

У підрозділі «Вимоги до інформаційної і програмної сумісності» повинно бути зазначено вимоги до інформаційних структур на вході і виході та методів розв'язання, вихідних кодів, мов програмування та програмних засобів, що використовуються програмою. За необхідності повинен забезпечуватися захист інформації та програм.

У підрозділі «Вимоги до маркування та упакування» у загальному випадку указують вимоги до маркування програмного виробу, варіанти та способи упакування.

У підрозділі «Вимоги до транспортування та збереження» повинні бути зазначені для програмного виробу умови транспортування, місця збереження, умови збереження, умови складування, терміни збереження в різних умовах.

У підрозділі «Спеціальні вимоги» у даній лабораторній роботі потрібно навести вимоги до графічного інтерфейсу користувача. Для цього потрібно виконати попереднє проектування інтерфейсу. Для таких цілей існує багато доволі простих і зручних програмних засобів, частина з яких є безкоштовними. Наприклад, програма Pencil надає набір інструментів для проектування інтерфейсів десктопних систем (рис. 1.1), мобільних додатків, веб-додатків.

У розділі «Вимоги до програмної документації» повинен бути зазначений попередній склад програмної документації і, за необхідності, спеціальні вимоги до неї.

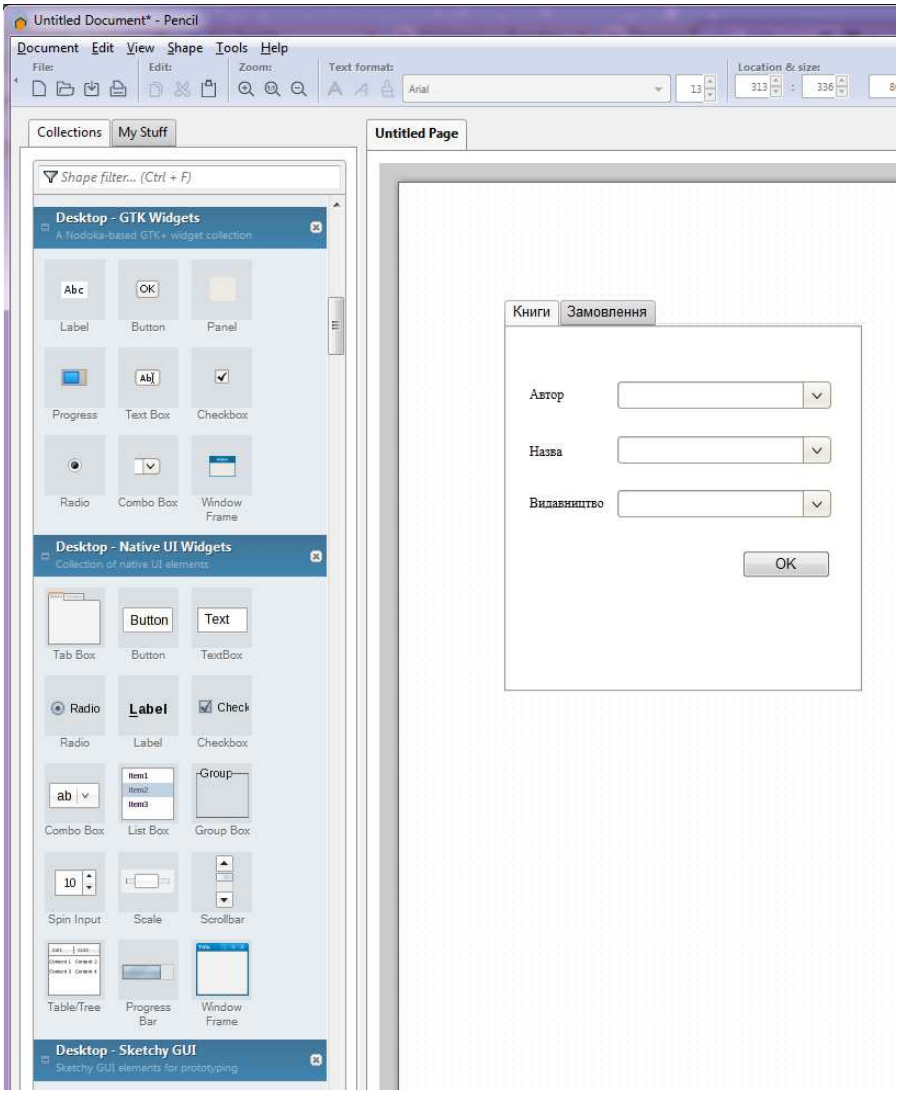

Рисунок 1.1 – Інструменти проектування інтерфейсів у програмі Pencil

У розділі «Техніко-економічні показники» повинні бути зазначені: орієнтована економічна ефективність, передбачувана річна пот-

реба, економічні переваги розробки порівняно з кращими вітчизняними та закордонними аналогами.

У розділі «Стадії та етапи розробки» установлюють необхідні стадії розроблення, етапи та зміст робіт (перелік програмних документів, що повинні бути розроблені, погоджені та затверджені), а також, як правило, терміни розроблення, та визначають виконавців.

У розділі «Порядок контролю та приймання» повинні бути зазначені види іспитів і загальні вимоги до приймання роботи. У даному розділі можуть такою зазначатися конкретні випробування, які мають проводитися для контролю та приймання програмного продукту.

1.3 Завдання до роботи

1.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для написання ТЗ на розробку програмного забезпечення.

1.3.2 Узгодити з викладачем індивідуальне завдання для розроблення програмного забезпечення. Необхідно враховувати, що отримана тема та відповідні завдання мають ґрунтуватися на наступних принципах: розроблювана у подальшому система має працювати з певним сховищем даних, має виконувати обробку та збереження даних, повинна забезпечуватися можливість користування великою кількістю різних груп користувачів, а також повинна мати візуальний користувацький інтерфейс.

1.3.3 Розробити ТЗ у відповідності до отриманого індивідуального завдання, а також відповідно до всіх вимог, які висуваються до ТЗ, та викладених теоретичних відомостей.

1.3.4 Провести критичний аналіз розробленого ТЗ на предмет відображення всіх вимог до розроблюваного програмного забезпечення та єдиного підходу до їх формулювання, а також на предмет правильно встановлених часових меж розробки. За необхідності внести корективи.

1.3.5 Оформити звіт з роботи.

1.3.6 Відповісти на контрольні запитання.

1.4 Зміст звіту

1.4.1 Мета роботи.

1.4.2 Предметна область, мета розроблення програмного забезпечення та задачі, які має виконувати розроблюваний програмний продукт.

1.4.3 Розроблене технічне завдання відповідно до теоретичних відомостей, предметної області, мети та завдань, що ставляться перед системою.

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

1.5 Контрольні запитання

1.5.1 З якою метою розробляється ТЗ?

1.5.2 Яку інформацію відображають у ТЗ?

1.5.3 Які основні складові, з яких має складатися ТЗ?

1.5.4 Як впливає якість розробленого ТЗ на кінцевий програмний продукт?

1.5.5 Хто має розробляти ТЗ?

1.5.6 Які засоби використовують для розроблення ТЗ?

1.5.7 Яким вимогам має відповідати ТЗ?

1.5.8 Якими особливостями характеризуються ТЗ у відповідності до особливостей розроблюваного програмного проекту?

1.5.9 Яку інформацію надає ТЗ виконавцям та замовнику?

1.5.10 Які стандарти визначають склад та вимоги до ТЗ?

2 ЛАБОРАТОРНА РОБОТА № 2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Мета роботи

Вивчити базові принципи та основи дослідження предметної області, у межах якої розробляється програмне забезпечення, і навчитися виконувати проектування архітектури системи на базі проведеного дослідження.

2.2 Короткі теоретичні відомості

Архітектура – це високорівнева частина проекту програмного додатку, каркас, що складається з деталей проекту. Як правило, архітектуру описують у єдиному документі, що називається «специфікацією архітектури».

У першу чергу специфікація архітектури повинна включати загальний опис системи.

У специфікації архітектури повинно міститися підтвердження того, що в процесі її розроблення були розглянуті альтернативні варіанти, і обґрунтований вибір остаточної організації системи. Описуючи альтернативні варіанти, обґрунтовується організація системи й показується, що роль кожного класу була ретельно розглянута.

Архітектура повинна визначати основні компоненти програми. В залежності від розміру програми її компонентами можуть бути окремі класи або підсистеми, що складаються з декількох класів. Кожний компонент є класом або набором класів/методів, які в сукупності реалізують високорівневі функції програми, такі як взаємодія з користувачем, відображення Web-сторінок, інтерпретація команд, інкапсуляція бізнес-правил або доступ до даних. У загальному випадку за кожну функцію додатка, зазначену у вимогах, повинен відповідати хоча б один компонент. Якщо функцію реалізують декілька компонентів, вони повинні співпрацювати, а не конфліктувати.

Архітектура повинна чітко визначати відповідальність кожного компонента. Компонент повинен мати одну область відповідальності і щонайменше знати про області відповідальності інших компонентів.

Архітектура повинна ясно визначати правила комунікації для кожного компонента. Вона повинна описувати, які інші компоненти даний компонент може використовувати безпосередньо, які опосередковано, а які взагалі не повинен використовувати.

Архітектура повинна визначати основні класи програмного додатка, їх області відповідальності й механізми взаємодії з іншими класами. Вона повинна описувати ієрархії класів, а також зміни станів і час існування об'єктів. Якщо система є досить великою, архітектура повинна описувати організацію класів у підсистеми.

Специфікація архітектури повинна обґрунтовано описувати основні види формату файлів і таблиць.

Архітектура, що залежить від специфічних бізнес-правил, повинна визначати їх і описувати їхній вплив на проект системи.

Архітектура повинна бути модульною, щоб графічний інтерфейс користувача можна було змінити, не торкаючись при цьому бізнес-правил і модулів програми, що відповідають за виведення даних. Наприклад, архітектура повинна забезпечувати можливість порівняно легкої заміни групи класів інтерактивного інтерфейсу на групу класів інтерфейсу командного рядка. Така можливість є досить корисною; багато в чому це пояснюється тим, що інтерфейс командного рядка зручний для тестування програмного забезпечення на рівні блоків або підсистем.

Специфікація архітектури повинна включати план керування обмеженими ресурсами, такими як з'єднання з базами даних, потоки та дескриптори. При розробці драйверів, вбудованих систем та інших додатків, які будуть працювати в умовах обмеженої пам'яті, необхідно також визначати спосіб керування пам'яттю. Необхідно визначити оцінку ресурсів, використовуваних у номінальному режимі роботи системи та за екстремального навантаження.

Архітектура повинна визначати підхід до безпеки на рівні проекту додатка та на рівні коду. Якщо модель потенційних небезпек дотепер не розроблена, це слід зробити при проектуванні архітектури. Про безпеку потрібно пам'ятати й при розробці принципів кодування, підходів до шифрування тощо.

У вимогах слід визначити показники продуктивності. Якщо вони пов'язані з використанням ресурсів, треба визначити пріоритети для різних ресурсів, у тому числі співвідношення швидкодії, використання пам'яті та витрат.

Специфікація повинна містити оцінки продуктивності й пояснювати, чому розробники архітектури вважають ці показники досяжними. Якщо вони можуть бути не досягнуті, це теж повинно бути відображено. Якщо для досягнення деяких показників необхідні специфічні алгоритми або типи даних, також потрібно це вказати в специфікації архітектури. Крім того, в архітектурі можна вказати обсяг простору й час, що виділяються кожному класу або об'єкту.

Масштабованістю називають можливість системи адаптуватися до росту вимог. Специфікація архітектури повинна описувати, як система буде реагувати на зростання числа користувачів, серверів, мережних вузлів, записів у базах даних, транзакцій тощо. Якщо розвиток системи не передбачається, а її масштабованість не відіграє важливої ролі, це повинно бути явно зазначене в архітектурі.

Якщо деякі дані або ресурси будуть загальними для розроблювальної системи й інших програм або обладнань, в специфікації архітектури потрібно вказати, як це буде реалізовано.

У процесі розроблення архітектури слід визначати схему зчитування даних та рівень, на якому будуть визначатися помилки введення-виведення: на рівні полів, записів, потоків даних або файлів.

При розробці архітектури системи слід вказати очікуваний рівень її відмовостійкості.

Специфікація архітектури повинна підтверджувати, що система є технічно здійсненною. Якщо неможливість реалізації якогось компонента може зробити проект непрацездатним, в архітектурі повинно бути відображено, як вивчалися ці питання: за допомогою прототипів, досліджень або іншим способом. Ці аспекти слід усунути до початку повномасштабного конструювання.

У специфікації архітектури повинне бути явно зазначено, чи можуть програмісти реалізувати у своїх блоках програми надлишкову функціональність або вони повинні створити найпростішу працездатну систему.

Якщо план передбачає застосування існуючого коду, тестів, форматів даних тощо, специфікація архітектури повинна пояснювати, як повторно використані ресурси будуть адаптовані до інших архітектурних особливостей, якщо це буде зроблено.

Розробникам архітектури програмного забезпечення слід зробити її досить гнучкою, щоб у систему можна було легко вносити ймовірні зміни.

2.3 Завдання до роботи

2.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

2.3.2 Провести аналіз предметної області, у рамках якої буде працювати розроблюване програмне забезпечення: виділити основні об'єкти, що зустрічаються у предметній області; виділити основні компоненти майбутньої системи; проаналізувати функції, які виконують виділені об'єкти предметної області.

2.3.3 На основі проведеного аналізу провести проектування архітектури розроблюваної системи. В результаті проведеного проектування має бути отриманий робочий варіант архітектури системи, що має включати: пакети, модулі, класи, бібліотеки тощо з описом призначення та функцій кожного з них.

2.3.4 Виходячи з отриманої архітектури та описаних у ТЗ вимог до системи, виконати обґрунтований вибір мови програмування. Мова програмування має надавати можливість використання об'єктно-орієнтованої парадигми програмування, а також має надавати можливість розроблення візуального інтерфейсу.

2.3.5 Виконати критичний аналіз отриманих результатів. За необхідності внести корективи у розроблену архітектуру або змінити обрану мову програмування.

2.3.6 Оформити звіт з роботи.

2.3.7 Відповісти на контрольні запитання.

2.4 Зміст звіту

2.4.1 Мета роботи.

2.4.2 Архітектура розроблюваного програмного проекту у відповідності до предметної області.

2.4.3 Аналіз розробленої архітектури. В аналізі мають бути надані відповіді на наступні запитання:

– чи ясно описана загальна організація програми? Чи включає специфікація грамотний огляд архітектури та її обґрунтування?

– чи адекватно визначено основні компоненти програми, їх області відповідальності й взаємодія з іншими компонентами?

– чи наведено опис найважливіших класів і їх обґрунтування?

– чи наведено опис організації даних і її обґрунтування?

– чи наведено опис організації й змісту сховища даних?

– чи визначені всі важливі бізнес-правила? Чи описано їх вплив на систему?

– чи описана стратегія проектування GUI?

– чи зроблено GUI модульним, щоб його зміни не впливали на іншу частину програми?

– чи наведено опис стратегії введення-виведення даних та її обґрунтування?

– чи вказано оцінки ступеню використання обмежених ресурсів, таких як потоки, з'єднання зі сховищем даних, дескриптори, пропускна спроможність мережі? Чи наведено опис стратегії керування такими ресурсами і її обґрунтування?

– чи описані вимоги до захищеності архітектури?

– чи визначає архітектура вимоги до обсягу й швидкодії всіх класів, підсистем і функціональних областей?

– чи описує архітектура спосіб досягнення масштабованості системи?

– чи розглянуті питання взаємодії системи з іншими системами?

– чи описана стратегія інтернаціоналізації/локалізації?

– чи визначена погоджена стратегія обробки помилок?

– чи визначений підхід до відмовостійкості системи (якщо це потрібно)?

– чи підтверджена можливість технічної реалізації всіх частин системи?

– чи визначений підхід до реалізації надлишкової функціональності?

– чи прийняті необхідні рішення відносно «покупки або створення» компонентів системи?

– чи описано у специфікації, як повторно використовуваний код буде адаптований до інших аспектів архітектури?

– чи зможе архітектура адаптуватися до ймовірних змін?

2.4.4 Обґрунтований вибір мови програмування.

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

2.5 Контрольні запитання

2.5.1 Що таке архітектура програмного продукту?

2.5.2 Яке місце займає проектування та розробка архітектури програмного забезпечення у життєвому циклі програмного забезпечення?

2.5.3 Які основні складові архітектури?

2.5.4 Яким вимогам має відповідати архітектура програмного забезпечення?

2.5.5 Як визначається якість архітектури програмного забезпечення?

2.5.6 Скільки часу (у відношенні до загального часу розробки) має розроблятися архітектура програмного продукту?

2.5.7 Яким чином можна описати архітектуру?

2.5.8 Які існують архітектурні шаблони?

2.5.9 Які існують основні принципи проектування архітектури?

2.5.10 Які існують архітектурні стилі?

3 ЛАБОРАТОРНА РОБОТА № 3 РЕАЛІЗАЦІЯ БАЗОВИХ ФУНКЦІЙ СИСТЕМИ

3.1 Мета роботи

Навчитися аналізувати технічне завдання, предметну область, на основі проведеного аналізу й дослідження виявляти першочергові для реалізації функції програмного забезпечення та реалізовувати їх.

3.2 Короткі теоретичні відомості

Для створення якісної та зручної програми, як з погляду користувача, так і розробників, які будуть продовжувати її розробку, необхідно дотримуватися принципів, наведених нижче.

1. На самому початку розробки, якщо стоїть вибір, наприклад, використовувати чи ні деяку бібліотеку/фреймворк/парадигму, необхідно зрозуміти, наскільки вона потрібна для розробки даного програмного забезпечення та яку частину її можливостей доведеться використовуватися в процесі розроблення, а також слід оцінити її вплив на розмір програмного додатка, швидкість його завантаження та зручність інсталяції або використання. Практика показує, що для невеликих програм використання великих фреймворків або закритих технологій не виправдовує себе відносно зручності для користувача.

2. Якщо додаток містить графічний інтерфейс, то потрібно ознайомитися з GUI-гайдлайнами для відповідної операційної системи.

3. Для того щоб забезпечити зручність інтерфейсу, рекомендується не реалізовувати ресурсомістких операцій в оброблювачах подій, особливо подій миші та подій зміни розмірів вікна. В оброблювачах подій повинен бути тільки мінімальний код, який буде запускати операції на фонове виконання, щоб у головній гілці додатка, що здійснює обробку всіх подій, не виникало великих черг.

4. У процесі розроблення програмного забезпечення треба не забувати про мету, поставлену для виконання проекту, – програма повинна виконувати саме те, що задумане спочатку, і нічого зайвого. Крім цього програма не повинна приводити до отримання некоректних результатів для коректних початкових даних. У випадку, якщо для розв'язання деякої підзадачі існує декілька способів, повинен обирати-

ся той, який буде давати правильну відповідь для будь-яких вхідних даних, навіть якщо це негативно впливає на продуктивність.

5. Незважаючи на те, що якісна програма повинна продуктивно працювати, це не означає, що потрібно оптимізувати програму з самого початку, до її написання або прямо в процесі.

6. Для розроблюваного додатка треба передбачити роботу з ним користувачів, що використовують пристрої з низькою продуктивністю, нестабільним каналом зв'язку або, наприклад, користувачів з мобільних пристроїв для веб-додатків, тобто для будь-яких умов необхідно забезпечити максимально комфортне використання програми.

7. Необхідно виконувати ретельне тестування релізів (і проміжних версій теж) продуктів у будь-яких ситуаціях.

8. Код програми має бути якісним, що включає вимоги оптимальності, наявності модульних тестів (юніт-тестів), зручності повторного використання, відповідності коду, який вважається стандартом у компанії, а також духу мови програмування. Заголовки файлів, публічні методи повинні обов'язково мати коментарі. Назви ідентифікаторів повинні бути осмисленими.

3.3 Завдання до роботи

3.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

3.3.2 Виконати аналіз ТЗ і розробленої архітектури системи та виділити першочергові функції, які має виконувати система.

3.3.3 Розробити програму, що реалізує першочергові функції, які має виконувати система. На даному етапі потрібно розробити консольний додаток (або декілька консольних додатків за необхідності). Реалізація програми виконується за допомогою обраної на попередньому етапі мови програмування. Реалізована програма має відповідати нормам програмування, тобто має бути виключена надлишковість програмного коду тощо, що має забезпечуватися за допомогою використання принципів об'єктно-орієнтованого програмування: поліморфізм, інкапсуляція, спадкування.

3.3.4 Виконати тестування розробленого програмного забезпечення. У процесі тестування має обов'язково застосовуватись модульне тестування.

Тестування має виконуватися за різноманітних умов роботи програми, тобто шляхом введення різних даних, шляхом виконання на пристроях з різними апаратними характеристиками, а також під керуванням різних операційних систем або версій операційних систем.

Результатами тестування є швидкість роботи програми, вимоги до ресурсів, а також коректність отримуваних результатів.

3.3.5 Виконати аналіз отриманих результатів тестування. У процесі аналізу отриманих результатів має бути порівняно результати, отримані під керування різних операційних систем (або їх версій) та на різних пристроях.

3.3.6 Оформити звіт з роботи.

3.3.7 Відповісти на контрольні запитання.

3.4 Зміст звіту

3.4.1 Мета роботи.

3.4.2 Перелік першочергових функцій, які має виконувати система, та перелік усіх інших, передбачених для реалізації, функцій з обґрунтуванням вибору.

3.4.3 Текст програми (з коментарями), що реалізує базові функції розроблюваного програмного продукту.

3.4.4 Результати тестування розробленого програмного забезпечення. Результати тестування мають відображатися у графічній та табличній формі для різних умов запуску.

3.4.5 Аналіз отриманих результатів тестування.

3.4.6 Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз. У висновках має бути відображено необхідність реалізації обраних базових функцій програмного продукту.

3.5 Контрольні запитання

3.5.1 Яким чином необхідно виконувати аналіз предметної області?

3.5.2 З яких міркувань слід виділяти основні, базові функції програмного забезпечення?

3.5.3 Як необхідно будувати ієрархію класів для можливості розширення у майбутньому?

3.5.4 Назвіть дії, що необхідно виконувати для майбутньої передачі коду.

3.5.5 Який код можна вважати якісним?

3.5.6 Яких принципів необхідно дотримуватися в процесі реалізації якісного програмного забезпечення?

3.5.7 Яким чином необхідно виконувати тестування для забезпечення якості програмного продукту?

3.5.8 У яких випадках необхідно використовувати готові рішення в процесі розроблення програмного забезпечення?

3.5.9 У чому полягає модульне тестування?

3.5.10 Яким чином використання об’єктно-орієнтованої парадигми програмування може впливати на якість програмного забезпечення?

4 ЛАБОРАТОРНА РОБОТА № 4 ВИКОРИСТАННЯ XML-ФАЙЛІВ ДЛЯ ЗБЕРЕЖЕННЯ ДАНИХ

4.1 Мета роботи

Ознайомитися з мовою XML та навчитися на практиці використовувати XML-файли в якості сховищ даних.

4.2 Короткі теоретичні відомості

XML (Extensible Markup Language) – розширювана мова розмітки, яка визначає множину правил кодування документів у форматі, зручному як для оброблення програмним чином, так і для людини.

Приклад XML-файлу:

```
<?xml version="1.0" encoding="utf-8"?>
<Orders>
  <Order>
    <Type>N</Type>
    <Date>Jan 1, 2015, 14:29</Date>
    <Customer>
      <SernaDirect>
        <SubscriptionType>B</SubscriptionType>
        <SubscriptionLength>12</SubscriptionLength>
      </SernaDirect>
      <Address>
        <Address1>123 Somewhere Ave.</Address1>
        <Address2></Address2>
        <City>Some Town</City>
        <State>TA</State>
        <Zip>000000000</Zip>
      </Address>
      <CreditCard>
        <Number>4111111111111111</Number>
        <CardHolderName>John Q Public</CardHolderName>
        <Expiry>11/09</Expiry>
```

```
    </CreditCard>
    <Phone>5555555555</Phone>
    <Name>John Public</Name>
    <Email>jpublic@someprovider.com</Email>
  </Customer>
  <ID>0000000001</ID>
  <Number>x582n9</Number>
  <Products>
    <Product>
      <Model>X9</Model>
      <Price>129.95</Price>
      <ID>x9000059</ID>
    </Product>
  </Products>
</Order>
<Order>
  <Type>N</Type>
  <Date>Jan 1, 2015, 16:00</Date>
  <Customer>
    <SernaDirect>
      <SubscriptionType>D</SubscriptionType>
      <SubscriptionLength>12</SubscriptionLength>
    </SernaDirect>
    <Address>
      <Address1>89 Subscriber's Street</Address1>
      <Address2>Box 882</Address2>
      <City>Smallville</City>
      <State>XQ</State>
      <Zip>000000000</Zip>
    </Address>
    <CreditCard>
      <Number>4512451245124512</Number>
      <CardHolderName>Helen
                          P Someperson</CardHolderName>
      <Expiry>01/08</Expiry>
    </CreditCard>
    <Phone>5554443333</Phone>
    <Name>Helen Someperson</Name>
```

```
      <Email>helens@isp.net</Email>
    </Customer>
    <ID>0000000002</ID>
    <Number>a98f78d</Number>
    <Products>
      <Product>
        <Model>Y9</Model>
        <Price>229.95</Price>
        <ID>y9000065</ID>
      </Product>
    </Products>
  </Order>
</Orders>
```

Об’єктна модель документів XML дозволяє завантажувати вміст XML-файлів у пам’ять, після чого з ними можна працювати, використовуючи властивості, методи та події об’єктної моделі документів.

У мові C# простір імен System.Xml надає підтримку оброблення XML, що ґрунтується на стандартах. Відповідно у програму необхідно додати наступний оператор:

```
using System.Xml;
```

Приклад читання вмісту елементів та атрибутів:

```
StringBuilder output = new StringBuilder()
String xmlString =
        @"<bookstore>
    <book genre='autobiography' publicationdate='1981-03-22'
ISBN='1-861003-11-0'>
        <title>The Autobiography of Benjamin Franklin</title>
        <author>
          <first-name>Benjamin</first-name>
          <last-name>Franklin</last-name>
        </author>
        <price>8.99</price>
      </book>
  </bookstore>";
```

```
using (XmlReader reader = XmlReader.Create(new StringReader(xmlString)))
{ reader.ReadToFollowing("book");
  reader.MoveToFirstAttribute();
  string genre = reader.Value;
  output.AppendLine("The genre value: " + genre);
  reader.ReadToFollowing("title");
  output.AppendLine("Content of the title element: " + reader.ReadElementContentAsString());
}
```

Для мови C++ існує багато XML-парсерів: Xerces-C++, libxml++, pugixml тощо.

4.3 Завдання до роботи

4.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

4.3.2 Виконати аналіз ТЗ та розробленої архітектури системи щодо вимог до організації даних.

4.3.3 Розробити структуру XML-файлів, необхідних для збереження даних системи.

4.3.4 Реалізувати функціональність програмного забезпечення, пов’язану зі взаємодією з даними.

4.3.5 Виконати тестування розробленого програмного забезпечення. У процесі тестування має обов’язково застосовуватись модульне тестування.

Тестування має виконуватися шляхом взаємодії з різними XML-файлами визначеної структури на пристроях з різними апаратними характеристиками під керуванням різних операційних систем або версій операційних систем.

4.3.6 Виконати аналіз отриманих результатів тестування. У процесі аналізу отриманих результатів має бути порівняно результати, отримані під керування різних операційних систем (або їх версій) та на різних пристроях.

4.3.7 Оформити звіт з роботи.

4.3.8 Відповісти на контрольні запитання.

4.4 Зміст звіту

4.4.1 Мета роботи.

4.4.2 Текст програми (з коментарями), що реалізує взаємодію зі сховищами даних.

4.4.3 Результати тестування розробленого програмного забезпечення з наведенням фрагментів відповідних XML-файлів.

4.4.4 Аналіз отриманих результатів тестування.

4.4.5 Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

4.5 Контрольні запитання

4.5.1 Що таке XML?

4.5.2 Для чого використовується мова XML?

4.5.3 Яким чином організована структура XML-файлів?

4.5.4 Який елемент документу XML називається кореневим?

4.5.5 Які існують кодування документів XML?

4.5.6 Які існують способи читання XML-файлів та чим вони відрізняються?

4.5.7 У чому полягає модель DOM?

4.5.8 У чому полягає модель SAX?

4.5.9 Що таке парсер?

4.5.10 Яким чином відбувається програмна взаємодія з XML-файлами для обраної Вами мови програмування?

5 ЛАБОРАТОРНА РОБОТА № 5 РОЗШИРЕННЯ ФУНКЦІОНАЛЬНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Мета роботи

Навчитися розширювати функціональність вже існуючого програмного забезпечення, розробленого іншими розробниками.

5.2 Короткі теоретичні відомості

Розроблення нової функціональності для існуючого програмного додатка є доволі складним завданням, оскільки завжди є відмінність між початковим дизайном та поточною реалізацією, а також між підходами до програмування різних програмістів або команд.

Для визначення поточного стану розробки додатка існують різні архітектурні інструменти, одними з яких є архітектурні інструменти в Visual Studio Team System 2010 або аналогічні засоби Visual Studio 2013, 2015, які дозволяють зрозуміти наявний додаток та прийняті в ньому рішення, спроектувати необхідну нову функціональність і перевірити відповідність реалізації проектуванню.

5.2.1 Визначення раніше розробленої функціональності

Визначення залежностей між частинами додатка може бути дуже важливим для пошуку потенційних проблем. До того ж, маючи візуальне представлення, можна знайти найкраще місце для впровадження нової функціональності в архітектурі системи.

У Visual Studio Team System 2010 можна одержати візуальне представлення рішень щодо збірки, просторів імен, класів або за налаштовуваним фільтром у вигляді документа в форматі Directed Graph Markup Language (DGML), використовуючи інструмент Generate Dependency Graph з пункту меню Architecture (рис. 5.1) та його відповідні підрозділи By Assembly, By Namespace, By Class, Custom…

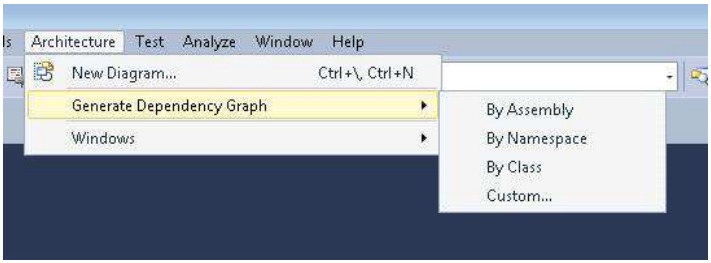

Рисунок 5.1 – Інструмент Generate Dependency Graph у Visual Studio 2010

Як продемонстровано на рис. 5.2, створений на підставі рішення DGML-документ може бути представлений у декількох варіантах: Dependency Matrix, Force Directed Layout або Top to Bottom Layout. Кожний варіант пропонує своє власне представлення структури проекту. Це високорівневе представлення, яке надає можливість оцінити загальне представлення архітектури додатка.

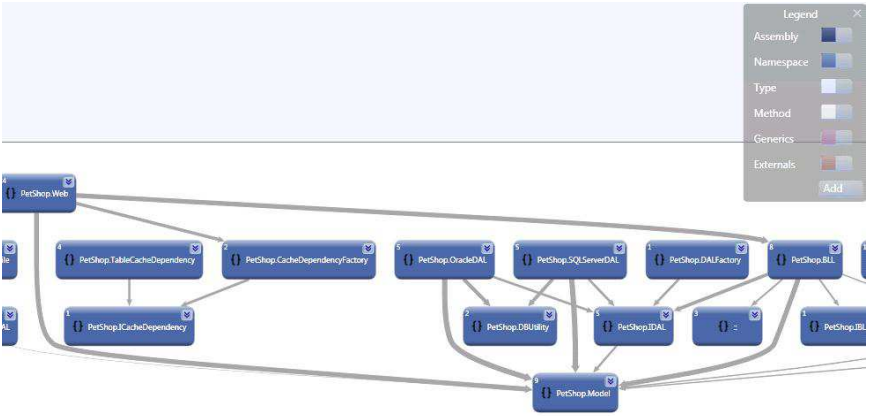

Рисунок 5.2 – DGML-документ

Візуалізація залежностей може бути дуже корисною під час аналізу. Наприклад, можливість візуалізувати взаємовідношення класу Cart з іншими класами дозволяє одержати дуже простий спосіб внести зміни в механізм кошика (рис. 5.3).

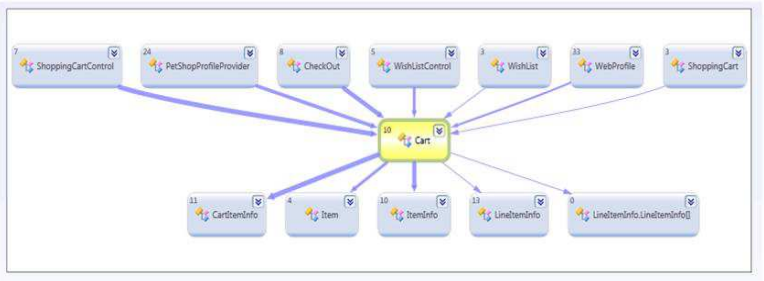

Рисунок 5.3 – Взаємовідносини класу Cart з іншими класами

Інший корисний спосіб зрозуміти, як працює додаток – це можливість візуалізації послідовності викликів, які відбуваються в ключових частинах додатка. Функція Generate Sequence Diagram, доступна через редактор коду, надає можливість спостерігати за викликами методів, які виконує додаток (рис. 5.4).

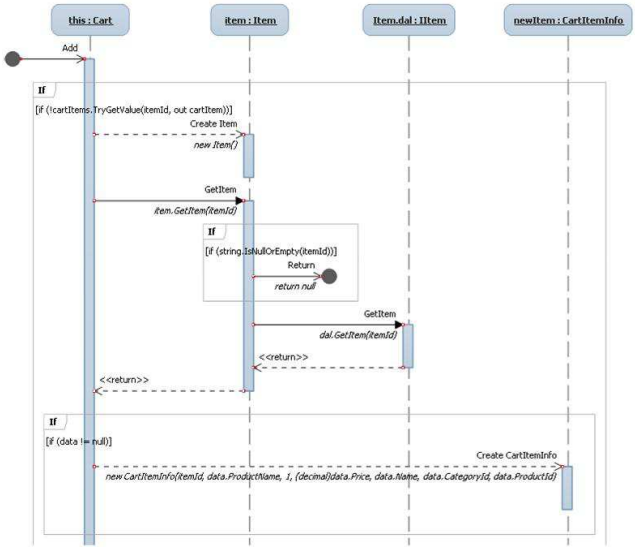

Рисунок 5.4 – Діаграма послідовностей

Аналогічні інструменти представлені в Visual Studio 2015 (рис. 5.5), але всі вони в даному випадку об’єднані під назвою мапи коду (Code Map).

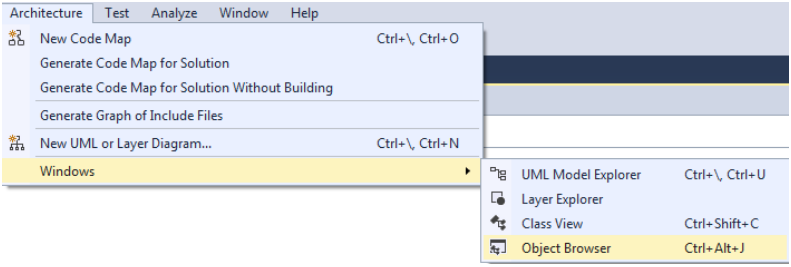

Рисунок 5.5 – Інструменти мап коду в Visual Studio 2015

Для швидкого розуміння архітектури проекту можна обрати з меню Architecture пункт Generate Code Map for Solution Without Building, що дозволить побудувати мапу, яка відображає граф структури рішення та посилання проекту, що дозволяє зрозуміти взаємозв'язки між різними частинами рішення. Після того, як всі бінарні файли будуть проіндексовані, мапа буде трансформована у представлення, наведене на рис. 5.6.

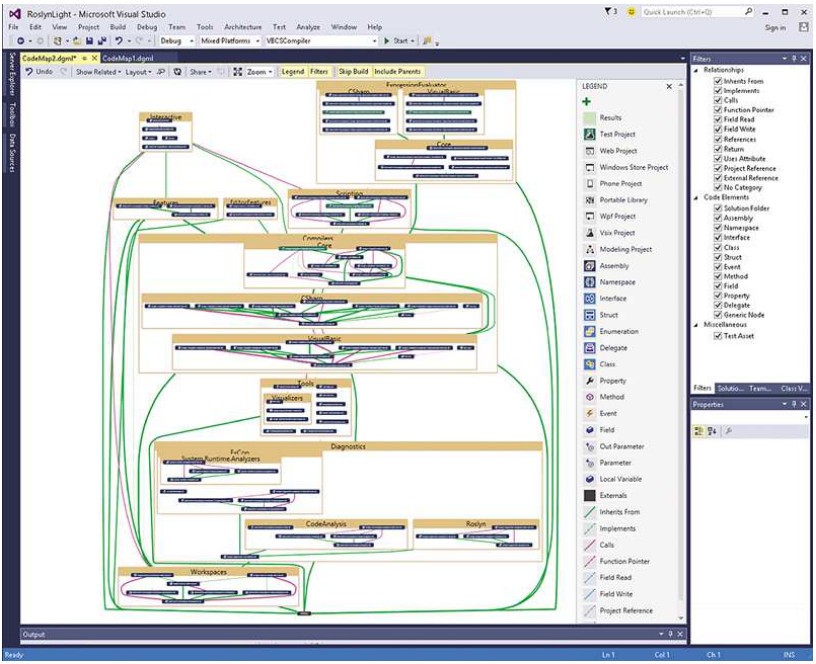

Рисунок 5.6 – Архітектурна мапа після індексування

На архітектурній мапі відображаються зв’язки, які зокрема демонструють наявність наслідування між класами в проектах (зелений колір ліній). Фільтри, розташовані справа, дозволяють виключити деякі залежності: наприклад, приховати код, який знаходиться на стадії тестування.

Можна збільшувати ділянки діаграми для подальшого дослідження або обирати вузол (вузли) для дослідження на іншій діаграмі, використовуючи пункт контекстного меню New Graph from Selection.

Для кожного програмного компоненту, що відображається на мапі, можна деталізувати простори імен, класи, розглядаючи залежності всередині кожного структурного блоку. Приклад такого відображення наведено на рис. 5.7 та на рис. 5.8.

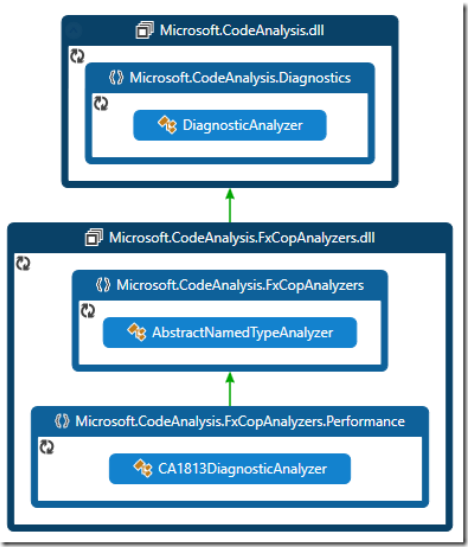

Рисунок 5.7 – Відображення базових класів для створеної мапи коду

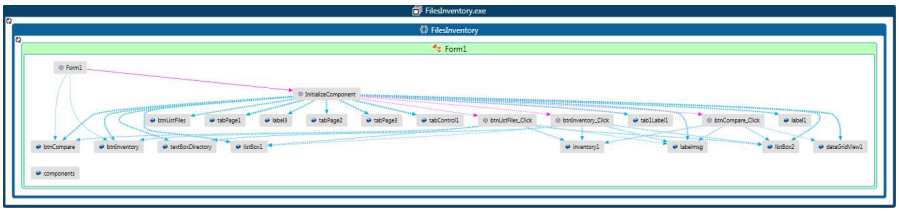

Рисунок 5.8 – Деталізоване відображення зв’язків на мапі коду

5.2.2 Проектування нової функціональності

Після того, як отримано більш-менш повне представлення про те, як працює існуючий додаток, можна більш ефективно розробити нову функціональність програмного забезпечення. Проектування вимагає стандартизації.

Діаграми Unified Modeling Language (UML) дозволяють представити структуру додатка в зрозумілому для інших вигляді. Наприклад, можна побудувати діаграми компонентів або діаграми класів UML, які описують існуючі елементи структури додатку, а потім додати в діаграми нові елементи, щоб проілюструвати й задокументувати зміни.

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

Для створення діаграми класів у Visual Studio 2015 необхідно обрати з меню Architecture пункт New UML or Layer Diagram…, що дозволить створити нову діаграму, обираючи відповідний тип (у даному випадку UML Class Diagram). Для додавання на діаграму існуючих типів необхідно з браузера рішення обрати відповідний клас та з контекстного меню обрати пункт View → View Class Diagram.

Приклад діаграми класів для розробленого програмного проекту наведено на рис. 5.9, де для кожного класу наведено перелік атрибутів та виконуваних методів.

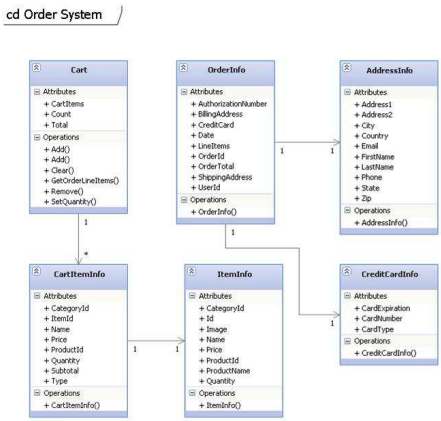

Рисунок 5.9 – Діаграма класів

Якщо на створеній діаграмі на будь-якому класі натиснути правою кнопкою (рис. 5.10), то до нього можна додати новий метод, властивість, поле, подію тощо, що призведе до автоматичного створення відповідного коду.

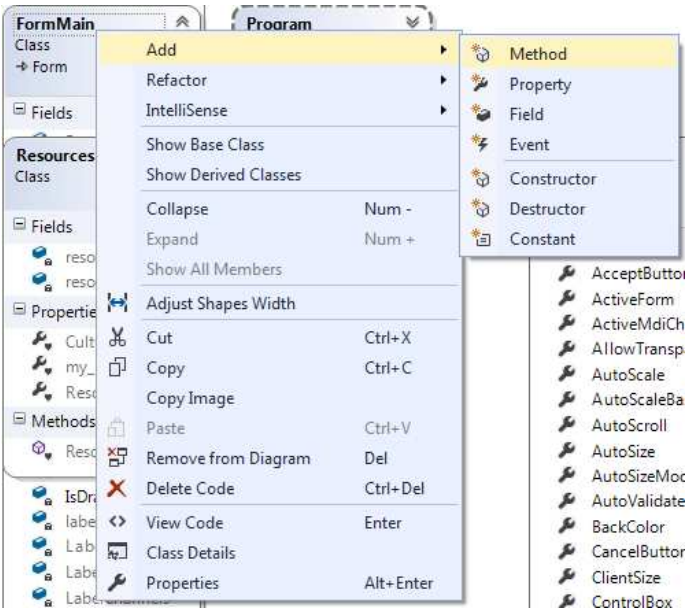

Рисунок 5.10 – Додавання нового методу до існуючого класу за допомогою діаграми класів

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

На рис. 5.11 представлено приклад такої діаграми компонентів для існуючого проекту.

Діаграма Use Case дозволяє краще зрозуміти функції й прийняти рішення щодо додавання нових функцій у додаток.

Діаграми Use Case застосовуються для моделювання вигляду системи з точки зору прецедентів (або варіантів використання). Даний

вид діаграм використовується для моделювання динамічних аспектів системи. Приклад побудови діаграми Use Case наведено на рис. 5.12.

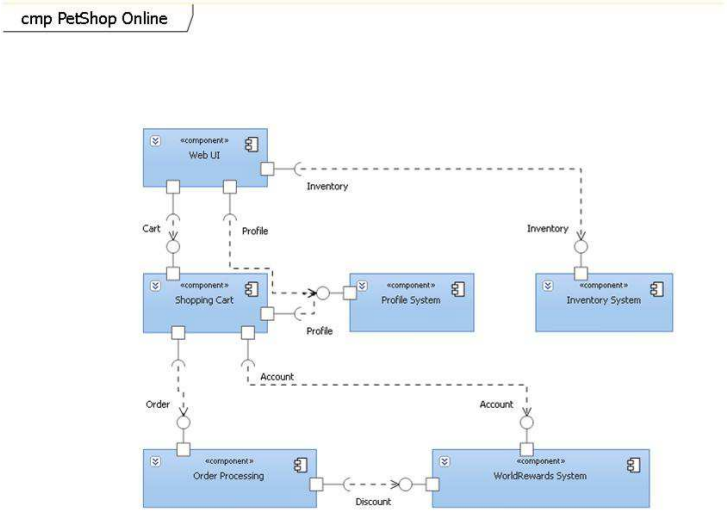

Рисунок 5.11 – Діаграма компонентів

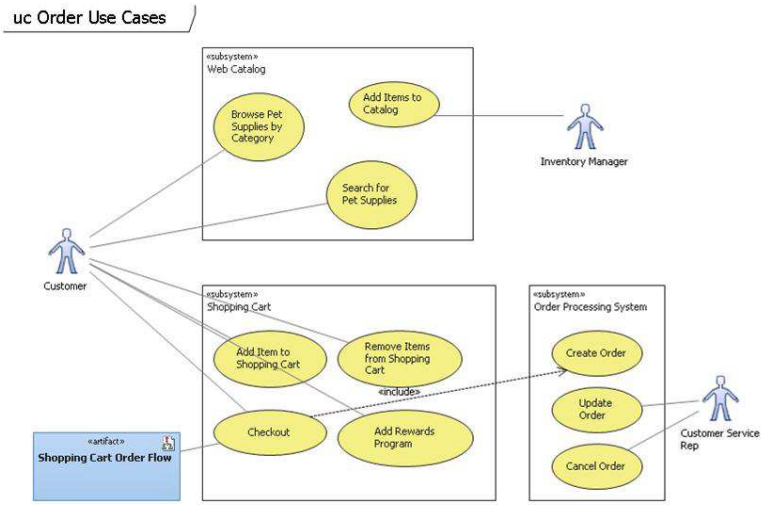

Рисунок 5.12 – Діаграма Use Case

Працюючи з діаграмами UML, можна створювати або зв'язувати елементи в дизайнері з елементами в системі Work Item Tracking сервера Team Foundation Server (TFS) (рис. 5.13).

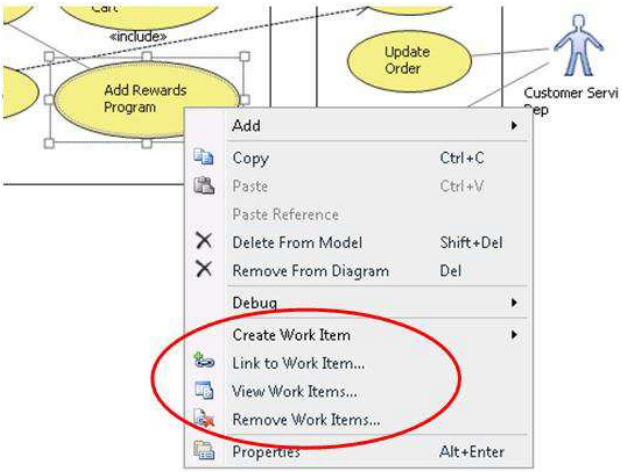

Рисунок 5.13 – Зв'язування елементів

5.3 Завдання до роботи

5.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

5.3.2 Узгодити з викладачем варіант індивідуального завдання, виконуваний іншим студентом, для виконання даної лабораторної роботи.

5.3.3 Одержати вихідні коди розробленого програмного забезпечення та розроблене технічне завдання.

5.3.4 Виконати аналіз архітектури системи, реалізованих функціональних характеристик та відповідність їх ТЗ. Виділити функції, що реалізовані у системі, та функції, ще не реалізовані у системі.

5.3.5 Реалізувати функції, що передбачені ТЗ та ще не реалізовані в отриманій програмній системі. При реалізації функцій необхідно дотримуватися принципу ідентичності, щоб нові функції принципово не відрізнялися від раніше розроблених.

5.3.6 Виконати тестування розробленого програмного забезпечення. У процесі тестування має обов'язково застосовуватись модульне тестування, направлене окремо на функції, розроблені власноруч, та на функції, розроблені попереднім розробником.

Тестування має виконуватися за різноманітних умов роботи програми, тобто шляхом введення різних даних, шляхом виконання на пристроях з різними апаратними характеристиками, а також під керуванням різних операційних систем або версій операційних систем.

Результатами тестування є швидкість роботи програми, вимоги до ресурсів, а також коректність отримуваних результатів.

5.3.7 Виконати аналіз отриманих результатів тестування. У процесі аналізу отриманих результатів має бути порівняно результати, отримані під керування різних операційних систем (або їх версій) та на різних пристроях.

5.3.8 Висунути пропозиції щодо розширення функціональності програмного забезпечення, орієнтуючись на розширення інноваційних характеристик програмного продукту.

5.3.9 Оформити звіт з роботи.

5.3.10 Відповісти на контрольні запитання.

5.4 Зміст звіту

5.4.1 Мета роботи.

5.4.2 Аналіз отриманого ТЗ та програмного забезпечення з обов'язковим наведенням діаграм і описом висновків.

5.4.3 Опис прийнятих рішень щодо розширення функціональності програмного забезпечення з використанням відповідних діаграм.

5.4.4 Текст програми (з коментарями), що реалізує розширення функціональності програмного продукту.

5.4.5 Результати тестування розробленого програмного продукту. Результати тестування мають відображатися у графічній та табличній формі для різних умов запуску.

5.4.6 Аналіз результатів тестування отриманої частини програмного забезпечення та власної розробки.

5.4.7 Пропозиції щодо розширення функціональності програмного забезпечення

5.4.8 Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз. У висновках має бути відо-

бражено функції, що вже були розроблено, та якість їх розроблення, необхідність розроблення нових функцій та прийняті щодо цього рішення, а також шляхи подальшого розвитку програмного проекту.

5.5 Контрольні запитання

5.5.1 Яким чином необхідно проводити аналіз ТЗ?

5.5.2 Яким вимогам мають відповідати назви змінних, класів тощо для адекватного розуміння змісту відповідних елементів іншими розробниками?

5.5.3 Як слід виконувати аналіз існуючої програмної розробки?

5.5.4 Які Ви знаєте засоби, що дозволяють полегшити процес аналізу існуючого програмного забезпечення?

5.5.5 Що таке графи залежностей та мапи коду?

5.5.6 Які файли мають формат DGML?

5.5.7 Яким чином побудувати діаграму класів та яким чином її можна застосувати під час розширення функціональності програмного додатку?

5.5.8 Яким чином побудувати діаграму компонентів та для чого її можна використати під час розробки?

5.5.9 Яким чином побудувати діаграму послідовностей і яка інформація на ній наводиться?

5.5.10 Для чого призначені діаграми Use Case та яким чином їх побудувати в існуючому програмному проекті?

6 ЛАБОРАТОРНА РОБОТА № 6 РЕАЛІЗАЦІЯ ІННОВАЦІЙНИХ ФУНКЦІЙ СИСТЕМИ

6.1 Мета роботи

Навчитися виділяти інноваційні функції системи, виконувати їх аналіз та реалізовувати на практиці.

6.2 Короткі теоретичні відомості

Сучасне програмне забезпечення часто належить до класу великих програмних проектів, під якими розуміють різновид інноваційного проекту, що розглядається як інтелектуальна діяльність колективу розробників, направлена на досягнення раніше визначеного результату / мети за умов заданих ресурсів та часових обмежень з врахуванням вимог до якості. Результатом такого програмного проекту є створення унікального програмного продукту.

Інноваційний проект може визначити як програму або загалом комплекс взаємопов'язаних програм, що забезпечують ефективне досягнення конкретної інноваційної мети, узгоджених за ресурсами, строками, виконавцями та документально оформлених.

Інноваційний процес включає організаційні, виробничі, технологічні, комерційні та інші заходи, що призводять до впровадження та розповсюдження інновацій.

6.3 Завдання до роботи

6.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

6.3.2 Одержати пропозиції щодо розширення інноваційної функціональності програмного забезпечення та виконати аналіз отриманих пропозицій.

6.3.3 Одержати вихідні коди розробленого програмного забезпечення.

6.3.4 Реалізувати інноваційні функції для розробленого програмного забезпечення.

6.3.5 Виконати тестування розробленого програмного забезпечення. У процесі тестування має обов'язково застосовуватись модульне тестування.

Тестування має виконуватися за різноманітних умов роботи програми, тобто шляхом введення різних даних, шляхом виконання на пристроях з різними апаратними характеристиками, а також під керуванням різних операційних систем або версій операційних систем.

6.3.6 Оформити звіт з роботи.

6.3.7 Відповісти на контрольні запитання.

6.4 Зміст звіту

6.4.1 Мета роботи.

6.4.2 Текст програми (з коментарями).

6.4.3 Результати тестування розробленого програмного забезпечення. Результати тестування мають відображатися у графічній та табличній формі для різних умов запуску.

6.4.4 Аналіз отриманих результатів тестування.

6.4.5 Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз. У висновках має бути відображено інноваційні функції, які було реалізовано, та виконано аналіз даного процесу.

6.5 Контрольні запитання

6.5.1 Який проект називають інноваційним?

6.5.2 З чого складається менеджмент інновацій?

6.5.3 Яким чином реалізація інноваційних функцій програмного продукту впливає на остаточний результат?

6.5.4 Яким чином інновації пов'язані з ризиками?

6.5.5 Що розуміють під стартапом?

7 ЛАБОРАТОРНА РОБОТА № 7 РОЗРОБЛЕННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ ПРОГРАМНОЇ СИСТЕМИ

7.1 Мета роботи

Узагальнити та поглибити на практиці знання та навички розроблення графічного інтерфейсу програмної системи.

7.2 Короткі теоретичні відомості

Якість користувацького інтерфейсу визначається такими поняттями як дизайн та зручність використання (юзабіліті, usability). Зручність використання та дизайн є важливими для кожної людини, що причетна до розробки. Пояснюється це тим, що метою практично будь-якого розробника є залучення потенційного користувача. Користувачу в свою чергу важливо, щоб досвід роботи з програмою був якомога більш позитивним (або найменш негативним у випадку примусу до роботи). Користувача не цікавлять внутрішні процеси, він їх не помічає. А значить, взаємодія з ним відбувається через інтерфейс, який повинен відповідати певним нормам. Для керування цими нормами і використовується поняття юзабіліті. Дизайн відіграє важливу роль у тому, щоб усе це якісно втілити в життя.

У підсумку основна мета – надати користувачеві якомога більш позитивний досвід під час взаємного обміну інформацією в процесі роботи з системою.

7.2.1 Дизайн

Стратегія побудови дизайну інтерфейсів програмного забезпечення загалом має ґрунтуватись на наступних рекомендаціях:

1. Видимість стану системи (правило зворотного зв'язку). Система повинна завжди інформувати користувача про стан своєї роботи за допомогою відповідних засобів у відповідну мить.

2. Поінформованість користувача. Користувач завжди повинен мати інформацію про поточний статус роботи програми.

3. Засоби забезпечення зворотного зв'язку. Вибір конкретного засобу зворотного зв'язку залежить від типу інформації, яку потрібно донести до користувача, а також від типу дії, що викликає потребу в зворотному зв'язку.

4. Час оповіщення. Проміжок часу, у який користувач одержує інформацію про реакцію на його дію або про якусь подію, повинен бути мінімальним. Це особливо важливо, оскільки від наявності або відсутності у користувача інформації про поточний стан системи залежать його подальші дії. Якщо він не буде знати, що остання операція була завершена невдало, то наступні дії можуть викликати нові помилки.

5. Відповідність системи реальному світу. Система повинна розмовляти з користувачем його мовою, тобто мають використовуватись поняття, образи і цілі концепції, які вже знайомі користувачу за реальним світом, з власного досвіду, до яких він звик. Представлення інформації та об'єктів у програмі повинно бути організовано в природньому й логічному порядку.

6. Свобода дій користувача. Користувач повинен мати контроль над системою й можливість змінити поточний стан програми.

7. Послідовність і стандарти. Принцип послідовності означає використання тих самих засобів для відображення подібних образів і виконання дій, що мають однакову природу. Принцип послідовності при розробці інтерфейсів повинен дотримуватися буквально у всьому.

8. Попередження помилок. Даний принцип широко розповсюджений і у звичайному житті та означає, що «Дизайн, який попереджає виникнення проблем, є кращим за найгарніше повідомлення про помилку».

9. Розуміння краще ніж запам'ятовування. При розробці інтерфейсу потрібно робити всі об'єкти, функції, дії видимими й легко доступними користувачу. Мінімізуйте запам'ятовування – користувач не повинен постійно намагатися тримати в пам'яті інформацію з однієї частини програми, щоб застосувати її в іншій. У будь-який момент користувачу повинно бути ясно, що потрібно робити в даний момент.

10. Гнучкість і ефективність використання. Правило цілком закономірне, оскільки програма в першу чергу повинна вирішувати завдання, над яким працює користувач. Однак при проектуванні інтерфейсу перед розроблювачем часто встає така проблема: потрібно, щоб інтерфейс був однаково зручний і для новачків, і для досвідчених

користувачів. Але потрібно враховувати те, що це багато в чому різні споживачі, з різними вимогами до програми й різним стилем роботи.

11. Естетичний і мінімалістичний дизайн. В інтерфейсі не повинно бути нічого зайвого. Не потрібно захаращувати інтерфейс програми елементами, які можуть бути недоречними та малокорисними. Кожний елемент інтерфейсу обов'язково відволікає частину уваги користувача. Це може привести до того, що видимість і, відповідно, легкість сприйняття користувачем дійсно потрібних і корисних частин інтерфейсу буде сильно зменшена за рахунок елементів, без яких у даному випадку можна було б цілком обійтися.

12. Розпізнавання й виправлення помилок. Допомагайте користувачу розпізнавати та виправляти помилки. Дане правило визначає проектування повідомлень про помилки. Гарні повідомлення про помилки – це повідомлення, які пояснюють, у чому полягає проблема й, найголовніше, як її виправити.

13. Опис помилки

14. Опис розв'язання проблеми. Інформація про те, як виправити помилку або розв'язати проблему, має навіть більше значення ніж сам опис помилки або проблеми, адже підказка, що допомагає розв'язати проблему, сприяє реалізації одного з принципів побудови користувацького інтерфейсу – програма повинна допомагати виконати завдання, а не ставати цим завданням.

15. Довідка та документація. Даний принцип полягає в необхідності надання користувачу довідкової системи та документації на програму.

Це п'ятнадцять головних заповідей будь-якого розробника комп'ютерних інтерфейсів, тобто мінімальні критерії, яким повинен відповідати інтерфейс будь-якої програми. Це п'ятнадцять евристичних правил найвідомішого американського фахівця в області проектування інтерфейсів Якоба Нільсена, розроблених ним разом з іншим дослідником, Рольфом Молічем.

7.2.2 Юзабіліті

Юзабіліті – поняття, що означає загальний рівень зручності об’єкта для використання в заявлених цілях. Термін має зв'язок з поняттям «ергономічність», але на відміну від останнього менше асоці-

ується з технічною естетикою, із зовнішнім виглядом і більш прив'язаний до утилітарності об'єкта, що використовується.

У більш вузькому представленні – це технологія, що впливає на всі етапи розробки програмної системи. Ґрунтується вона на особливостях психологічного сприйняття інформації людським мозком. Основна її ціль – проконтролювати, спрогнозувати й впливати на процеси розроблення системи з метою підвищення кінцевої ергономічності продукту.

Юзабіліті-технологія застосовується на всіх етапах розробки. Розглянемо основні правила, які використовуються під час застосування даної технології в процесі розробки.

Правило 7±2. Можливості мозку з обробки інформації не безмежні. Відповідно до результатів дослідження Джорджа Міллера, короткочасна пам'ять може одночасно містити від 5 до 9 сутностей. Цей факт часто використовується під час обґрунтування необхідності скоротити кількість елементів у навігаційному меню до 7.

Правило 2-х секунд. Полягає в тому, що користувач не повинен чекати певної реакції системи, наприклад, запуску або перемикання додатка, більше 2-х секунд. У загальному випадку, чим менше чекає користувач, тим краще.

Правило 80/20 (Принцип Парето). Передбачає, що 80% ефекту виходить у результаті 20% дій. У бізнесі це правило часто застосовується у вигляді: «80% продажів припадає на 20% клієнтів».

Правило Фіттса. Опублікована Паулем Фіттсом у 1954 році модель рухів людини визначає час, необхідний для швидкого переміщення в цільову зону як функцію від відстані до мети й розміру мети. Зазвичай це правило використовується при розгляді руху мишею від точки А до токи В. Це може бути важливо при розміщенні елементів, кількість кліків на які бажано збільшити.

Задоволеність. Користувачі не вибирають оптимальний шлях у пошуках необхідної інформації. Їм не потрібен найкращий і найнадійніший розв'язок, часто вони навпаки готові задовольнитися швидким і не найкращим розв'язком, який буде «цілком прийнятним».

7.3 Завдання до роботи

7.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

7.3.2 Провести аналіз ТЗ на предмет вимог щодо візуального інтерфейсу програмного забезпечення.

7.3.3 Розробити візуальний інтерфейс у відповідності до вимог у ТЗ та попереднього проектування інтерфейсу в ТЗ.

7.3.4 Виконати аналіз розробленого інтерфейсу відповідно до відомих вимог до інтерфейсів. Обґрунтувати прийняті рішення, базуючись на відомих правилах, законах, принципах, характеристиках.

7.3.4 Виконати тестування розробленого програмного забезпечення. Тестування має виконуватися на непідготовленому користувачі за різноманітних умов роботи програми, тобто шляхом введення різних даних, шляхом виконання на пристроях з різними апаратними характеристиками, а також під керуванням різних операційних систем або версій операційних систем та з різними параметрами налаштування монітору. Результатами тестування є швидкість роботи програми, вимоги до ресурсів, а також коректність отримуваних результатів та адекватність відображення інтерфейсу програми і взаємодії з користувачем.

7.3.5 Виконати аналіз отриманих результатів тестування. У процесі аналізу отриманих результатів має бути порівняно результати, отримані за різних умов виконання програми.

7.3.6 Оформити звіт з роботи.

7.3.7 Відповісти на контрольні запитання.

7.4 Зміст звіту

7.4.1 Мета роботи.

7.4.2 Аналіз ТЗ та розробленого програмного забезпечення щодо вимог до користувацького інтерфейсу з наведенням попередньо спроектованого в ТЗ варіанту.

7.4.3 Результати розроблення користувацького інтерфейсу програмного забезпечення у вигляді відповідних форм.

7.4.4 Обґрунтування прийнятих рішень щодо інтерфейсу програмного забезпечення.

7.4.5 Текст програми (із коментарями), що реалізує користувацький інтерфейс.

7.4.6 Результати тестування розробленого програмного продукту. Результати тестування мають відображатися у графічній та табличній формі для різних умов запуску.

7.4.7 Аналіз отриманих результатів тестування.

7.4.8 Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

7.5 Контрольні запитання

7.5.1 Що розуміють під терміном юзабіліті?

7.5.2 Що розуміють під терміном дизайн?

7.5.3 Що розуміють під терміном інтерфейс та які існують види інтерфейсу?

7.5.4 Яке значення займають юзабіліті та дизайн при розробці користувацького інтерфейсу програмного забезпечення?

7.5.5 Виділіть рекомендації, у відповідності до яких слід розробляти дизайн користувацького інтерфейсу.

7.5.6 Назвіть основні правила юзабіліті.

7.5.7 Назвіть основні засоби проектування користувацького інтерфейсу.

7.5.8 У чому полягає правило Фіттса?

7.5.9 Яким чином застосовується принцип Парето під час проектування інтерфейсу?

7.5.10 Яким чином визначити взаємне розташування елементів інтерфейсу?

8 ЛАБОРАТОРНА РОБОТА № 8 ОФОРМЛЕННЯ АВТОРСЬКОГО СВІДОЦТВА НА РОЗРОБЛЕНУ ПРОГРАМУ

8.1 Мета роботи

Засвоїти основні поняття про порядок та особливості оформлення авторського свідоцтва в Україні на розроблену програму, навчитися виділяти та представляти отримані результати.

8.2 Короткі теоретичні відомості

Відповідно до статті 18 Закону України «Про авторське право і суміжні права» в редакції від 5 грудня 2012 року комп'ютерні програми охороняються як літературні твори. Така охорона поширюється на комп'ютерні програми незалежно від способу чи форми їх вираження.

Приклад порядку реєстрації авторського права на комп'ютерну програму в Україні можна навести у вигляді послідовності наступних етапів:

1. Заявником заповнюється і подається (особисто, поштою чи використовуючи e-mail) Патентному повіреному заявка, де вказуються його реквізити і описується суть предмету захисту – твору (програмне забезпечення, база даних).

2. Патентний повірений проводить регіональну реєстрацію поданої заявки, розрахунок витрат на державну реєстрацію права на твір, оформлює кошторис витрат з проектом договору на виконання робіт з реєстрації авторського права.

3. Заявник подає Патентному повіреному підписаний договір, кошторис, довіреність, текст твору і додаткові матеріали (за наявності), що необхідні для опису вказаних об'єктів.

4. Заявник оплачує Патентному повіреному передбачений договором аванс – 50% суми договору.

5. Патентний повірений виконує роботи з дослідження предмета захисту та оформлення заяви і документів на державну реєстрацію авторського права на твір (*протягом одного місяця за звичайним тарифом):*

– аналіз предмету захисту і визначення системи захисту його складових;

– визначення форми подання твору на реєстрацію та його оформлення;

– оформлення опису щодо використання твору та його структури;

– оформлення інших документів відповідно до «Порядку державної реєстрації авторського права і договорів, які стосуються права автора на твір».

6. Заявник підписує заявочні документи і оплачує Патентному повіреному 30% суми договору.

7. Патентний повірений оплачує передбачений в кошторисі державний збір за підготовку до реєстрації авторського права на твір і особисто подає заявку в Українське агентство авторських і суміжних прав (УААСП) Державного департаменту інтелектуальної власності на реєстрацію авторського права на твір.

8. Патентний повірений супроводжує формальну експертизу УААСП заявки і приймає певні рішення чи надає відповідні додаткові документи на запити експерта (*протягом одного місяця*).

9. УААСП установлює пріоритет заявки і видає Патентному повіреному рішення про реєстрацію авторського права на твір (*протягом місяця)*.

10. Заявник оплачує Патентному повіреному 20% суми договору, який оплачує державне мито за видачу свідоцтва на авторське право (згідно Декрету КбМУ «Про державне мито»).

11. УААСП заносить відомості про реєстрацію до Державного реєстру свідоцтв, публікує інформацію про зареєстроване авторське право в офіційному бюлетені «Авторське право і суміжні права» і видає Патентному повіреному свідоцтво України на авторське право (*протягом одного місяця*).

12. Патентний повірений вручає свідоцтво України на авторське право Заявнику і надає консультації щодо його правового супроводження, в т.ч. щодо оформлення авторських договорів.

Державна реєстрація авторського права і договорів здійснюється відповідно до Закону України «Про авторське право і суміжні права» і Постанови КбМУ «Про державну реєстрацію авторського права і договорів, які стосуються права автора на твір».

Заявка на реєстрацію авторського права на твір повинна обов’язково містити:

– заяву (викладену українською мовою, що складається за встановленою формою);

– примірник твору;

– документ, що свідчить про факт і дату оприлюднення твору (за наявності);

– документ або копію документа про сплату збору за підготовку до реєстрації авторського права, або копію документа, що підтверджує наявність пільг;

– документ про сплату збору за оформлення і видачу свідоцтва або копію документа, що підтверджує наявність пільг;

– довіреність, оформлену в установленому порядку, якщо заявка подається довіреною особою.

8.3 Завдання до роботи

8.3.1 Ознайомитися з літературою та основними теоретичними відомостями, необхідними для виконання роботи.

8.3.2 Розглянути розроблену програмну систему на предмет відповідності поставленому ТЗ.

8.3.3 Оформити настанову щодо використання створеного програмного продукту, що має складатися з наступних елементів:

– призначення програми;

– структура системи;

– функціонування системи;

– структура та організація даних;

– основні функції;

– інтерфейсні засоби;

– вимоги до програмного та апаратного забезпечення;

– методика роботи користувача з системою.

8.3.4 Оформити заяву про реєстрацію авторського права на твір.

8.3.5 Оформити договір між роботодавцем та авторами комп’ютерної програми.

8.3.6 Створити презентацію для представлення результатів розробленого проекту.

8.3.7 Оформити звіт з роботи.

8.3.8 Відповісти на контрольні запитання.

8.4 Зміст звіту

8.4.1 Мета роботи.

8.4.2 Аналіз ТЗ та розробленого програмного забезпечення.

8.4.3 Настанова щодо використання розробленої програмної системи.

8.4.4 Заява та договір.

8.4.5 Фрагмент тексту розробленої програмної системи.

8.4.6 Презентація програмного продукту.

8.4.7 Висновки, що відображають особисто отримані результати виконання роботи та їх критичний аналіз.

8.5 Контрольні запитання

8.5.1 Які законодавчі акти регламентують оформлення авторського свідоцтва на твір?

8.5.2 Хто має право на оформлення авторського свідоцтва на програму?

8.5.3 Чим є з законодавчої точки зору комп'ютерна програма?

8.5.4 Чим є з законодавчої точки зору база даних?

8.5.5 Які документи необхідно оформити для реєстрації авторського права на програму?

8.5.6 Хто займається оформленням документів під час реєстрації авторського права на програму?

8.5.7 Опишіть порядок реєстрації авторського свідоцтва на комп'ютерну програму.

8.5.8 Які органи державної влади приймають участь у процесі реєстрації та оформлення авторського права на програму?

8.5.9 У які строки приймається рішення щодо надання авторського свідоцтва на комп'ютерну програму?

8.5.10 Що представляє собою настанова щодо використання комп'ютерної програми?

ЛІТЕРАТУРА

1. Липаев, В. В. Программная инженерия. Методологические основы : учеб. пособие [Текст] / В. В. Липаев. – М. : ТЕИС, 2006. – 608 с.

2. Штерн, В. Основы С++. Методы программной инженерии [Текст] / В. Штерн. – М. : Лори, 2003. – 880 с.

3. МакКоннелл, С. Совершенный код. Практическое руководство по разработке программного обеспечения [Текст] / С. МакКоннелл. – СПб. : Питер, 2007. – 896 с.

4. Соммервилл, И. Инженерия программного обеспечения / И. Соммервилл. – М. : Вильямс, 2002. – 624 с.

5. Павлова, В. Л. Рекомендации по преподаванию программной инженерии и информатики в университетах [Текст] : пер. с англ. / В. Л. Павлова, А. А. Терехова, А. Н. Терехова. – М. : ИНТУИТ.РУ, 2007. – 462 с.

6. ГОСТ 19.101-77. Виды программ и программных документов [Текст]. – Введ. 1980-01-01. – М. : Изд-во стандартов, 1987. – 15 с.

7. ISO/IEC 12207:2008. Systems and software engineering – Software Life Cycle Processes [Текст]. – Second edition 2008-02-01. – Switzerland : International Organization for Standardization. – 138 p.

8. Раскин, Дж. Интерфейс : новые направления в проектировании компьютерных систем : Пер. с англ. [Текст] / Джеф Раскин. – СПб. : Символ-Плюс, 2004. – 272 с.

9. Вигерс, К.И. Разработка требований к программному обеспечению : Пер, с англ. [Текст] / К.И. Вигерс. – М. : Издательско-торговый дом "Русская Редакция", 2004. – 576с. : ил.

10. Мацяшек, Л.А. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML : Пер. с англ. [Текст] / Л.А. Мацяшек. – М. : Издательский дом “Вильямс”, 2002. – 432 с., ил.

11. Vliet, H. Software Engineering [Текст] : Principles and Practice / Hans van Vliet ; 3rd Edition. – Wiley. – 2008. – 740 p.

12. Буч, Г. Язык UML. Руководство пользователя [Текст] / Гради Буч, Джеймс Рамбо, Ивар Якобсон. – М. : ДМК Пресс, 2006. – 496 с.

13. Леоненков, А.В. Самоучитель UML. 2-е издание [Текст] / А.В. Леоненков. – СПб. : "БХВ – Петербург", 2004. – 432 с.

14. Фаулер, М. Архитектура корпоративных программных приложений [Текст] : пер. с англ. / Мартин Фаулер. – М. : Издательский дом “Вильямс”, 2006. – 544 с. : ил. – Парал. тит. англ.

15. Эванс, Э. Предметно-ориентированное проектирование (DDD) [Текст] : структуризация сложных программных систем : пер. с англ. / Эрик Эванс. – М. : ООО “И.Д. Вильямс”, 2011. – 448 с. : ил. – Парал. тит. англ.

16. Про авторське право і суміжні права [Електронний ресурс] : Закон України в редакції від 5 грудня 2012 року. – Режим доступу : http://zakon4.rada.gov.ua/laws/show/3792-12

17. Про державну реєстрацію авторського права і договорів, які стосуються права автора на твір [Електронний ресурс] : Постанова Кабінету Міністрів України від 27 грудня 2001 р. № 1756 в редакції від 21.09.2011. – Режим доступу : http://zakon2.rada.gov.ua/laws/main/1756-2001-%D0%BF/print1198759817852676

18. Анализ и моделирование архитектуры [Електронний ресурс] / Режим доступу : https://msdn.microsoft.com/ru-ru/library/57b85fsc.aspx

19. Professional Visual Studio 2010 [Текст] / Nick Randolph, David Gardner, Chris Anderson, Michael Minutillo. – Wrox. – 2010. – 1224 p.

20. Beginning XML [Текст] / David Hunter, Jeff Rafter, Joe Fawcet etc. ; 4 edition. – Wrox. – 2007. – 1080.