ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської кваліфікаційної роботи

на ступінь вищої освіти бакалавр

на тему: **«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ПРОЕКТАМИ ТА ПРОЦЕСАМИ ЗА ДОПОМОГОЮ МОВИ C#»**

Виконав: студент 4 курсу, групи ПД-44

спеціальності 121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Сербулов Н.Є.

(прізвище та ініціали)

Керівник Гаманюк І.М.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Нормоконтроль

(прізвище та ініціали)

Київ – 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність -121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного
забезпечення
_____О.В. Негоденко
«____»_____2021 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

СЕРБУЛОВУ НІКІТІ ЄВГЕНІЙОВИЧУ

1.Тема роботи: «Розробка програмного забезпечення для управління проектами та процесами за допомогою мови C#»

Керівник роботи Гаманюк Ігор Михайлович, старший викладач.

2. Строк подання студентом роботи 01.06.2021

3. Вхідні дані до роботи:

3.1. Середовище розробки Visual Studio 2019

3.2. Алгоритм дії програми

3.3. Windows Presentation Foundation, .NET Framework

3.4. Науково-технічна література, пов’язана з управлінням проектами за допомогою різних методологій

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1. Аналіз обов’язків розроблюваної програми

4.2. Аналіз та порівняння існуючих прототипів

4.3. Дослідження програмних засобів для розробки програми

4.4. Розробити функціонал створеної програми

5. Перелік графічного матеріалу

5.1.1. Методології управління проектами

5.1.2. Переваги та недоліки популярних прототипів

5.1.3. Програмні засоби реалізації

5.1.4. Огляд можливостей програми

5.1.5. Апробація результатів досліджень

6. Дата видачі завдання 19.04.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	19.04.21 – 22.04.21	
2	Аналіз існуючих прототипів	22.04.21 – 23.04.21	
3	Дослідження програмних засобів	23.04.21 – 01.05.21	
4	Моделювання об’єкту проектування	01.05.21 – 05.05.21	
5	Розробка функціоналу бота	05.05.21 – 07.05.21	
6	Вступ, висновки, реферат	07.05.21 – 08.05.21	
7	Розробка презентації застосунку	08.05.21 – 24.05.21	
8	Попередній захист роботи	25.05.21	

Студент

Керівник роботи

РЕФЕРАТ

Текстова частина бакалаврської роботи 44с., 16 рис., 18 джерел.

Ключеві слова: Kanbab, C#, PM, Visual Studio, Agile, дошка, проект, менеджер проекту.

Об'єкт дослідження – поліпшення роботи в команді, та збереження часу на планування проектів.

Предмет дослідження – програмний продукт, для управління проектами та процесами.

Мета роботи – Розробка програмного забезпечення для управління проектами та процесами за допомогою мови C#.

Методи дослідження – методи теорії інформації, методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування, валідації та верифікації програмного забезпечення.

Наукова новизна даної роботи полягає в наступному:

1. Розроблено програму для поліпшення роботи команди програмістів.
2. Встановлено, що Kanban дошка це найкращий на даний момент система для управління процесами.
3. На основі результатів виконаних порівнянь та досліджень, було розроблено авторську програму для управління проектів.

В роботі виконано аналіз існуючих програм аналогів. Встановлено переваги та недоліки існуючих програм. В результаті аналізу було визначено основні потреби користувачів. Проаналізовано можливості середовища розробки Visual Studio. Розроблено логіку практичних завдань та загальну концепцію представлення інформації для користувачів.

Галузь використання – програму може використовувати будь-яка людина, яка є користувачем комп'ютера, та знається в основах проектного менеджменту.

ЗМІСТ

ВСТУП

Кожного ранку ми плануємо наші справи на цілий день. В програмуванні все так само. Люди в IT сфері завжди планують свою роботу на цілий день, а то й більше. Всім відомо, як швидко розвивається економіка в країнах Європи і яких масштабів вона досягла останнім часом. Все це завдяки ефективному плануванню, а також високої працездатності населення.

Щоденне планування завжди починається з призначення завдань, для яких встановлений час на робочі та вихідні дні. Коли все це переноситься в програму-календар або щоденник, ви можете побачити таку закономірність. Виявляється, у вас більше вільного часу, ніж ви собі уявляєте, і кожного дня у вас є шанс зробити набагато більше.

Актуальність даної роботи полягає в тому, що її можна використовувати у різних напрямках — це може бути або будівництво або впроваджування програмного забезпечення, опис документів чи розробка статті.

Метою випускної дипломної роботи — є розробка програмного забезпечення для управління проектами та процесами за допомогою мови C#.

Були поставлені наступні завдання, базуючись на встановленій меті:

- аналіз обраної предметної області;
- порівняння наявних аналогів;
- вибір технологій і середовища розробки;
- розробка програми за допомогою мови C#.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1.1 Опис предметної області

1.1.1 Основні відомості гнучких методологій

Гнучкі методології — це підходи до розробки продуктів, які відповідають цінностям і принципам, описаним в Agile Manifesto для розробки програмного забезпечення. Гнучкі методології, націлені на надання правильного продукту, з поступовою і частою доставкою невеликих фрагментів функціональності, через невеликі крос-функціональні самоорганізуються групи, що дозволяє часто отримувати зворотний зв'язок з клієнтами і коригувати курс у міру необхідності.

Застосування гнучкої методології продовжувалося протягом більшої частини своєї короткої історії (з 1999-2000 рр.) «Agile» переважно підходив до розробки програмного забезпечення та проектів з розробки IT-додатків. Однак, з того часу він тепер поширюється і на інші сфери, особливо в галузі знань та сфери послуг.

Agile — це швидке реагування на ринок та на споживача, швидко задовольняючи їх потреби та вимоги, а також мати можливість змінити напрямок, як того вимагає ситуація. Будь то IT, чи розробка програмного забезпечення, або будь-яка інша сфера, де є потік роботи та постачання робочих продуктів, застосовуються гнучкі методи. Швидкі методи намагаються максимізувати доставку вартості споживачеві та мінімізують ризик побудови продуктів, які не відповідають — або вже не відповідають потребам ринку чи споживача.

Вони роблять це, розбиваючи традиційно довгий цикл доставки (типовий для застарілих «методів водоспаду») на більш короткі періоди, які називаються спринтами або ітераціями. Ітерація забезпечує каденцію для доставки робочого продукту замовнику, отримання зворотного зв'язку та внесення змін на основі зворотного зв'язку.

Таким чином, Agile методи прагнули скоротити терміни доставки (доставляти раніше, доставляти часто), щоб забезпечити потрапляння на ринок менших вертикальних шматків товару, що дозволяє клієнтам надавати зворотній зв'язок на ранній стадії та гарантувати, що товар, який вони нарешті отримують, відповідає їхнім потребам.

Agile став загальним терміном для різноманітних методів і процесів планування, управління та технічних процесів для управління проектами, розробки програмного забезпечення та інших продуктів та послуг в ітераційному порядку. До цих методів належать Scrum, безумовно, найпоширеніший і найпопулярніший метод програмного забезпечення, XP (екстремальне програмування або парне програмування) та останнім часом Kanban.

До гнучких методів належать також технічні практики — більшість з яких підпадає під загальний термін DevOps — які дозволяють автоматизувати тести, постійну інтеграцію, безперервну доставку, розгортання і загалом постійно скорочується цикл доставки програмного забезпечення та інших продуктів та послуг.

Використання Agile як підходу до управління проектами різко зросло за останні роки. Гартнер прогнозує, що швидкі методи розробки незабаром будуть використані у 80% усіх проектів з розробки програмного забезпечення зображено на рисунку 1.

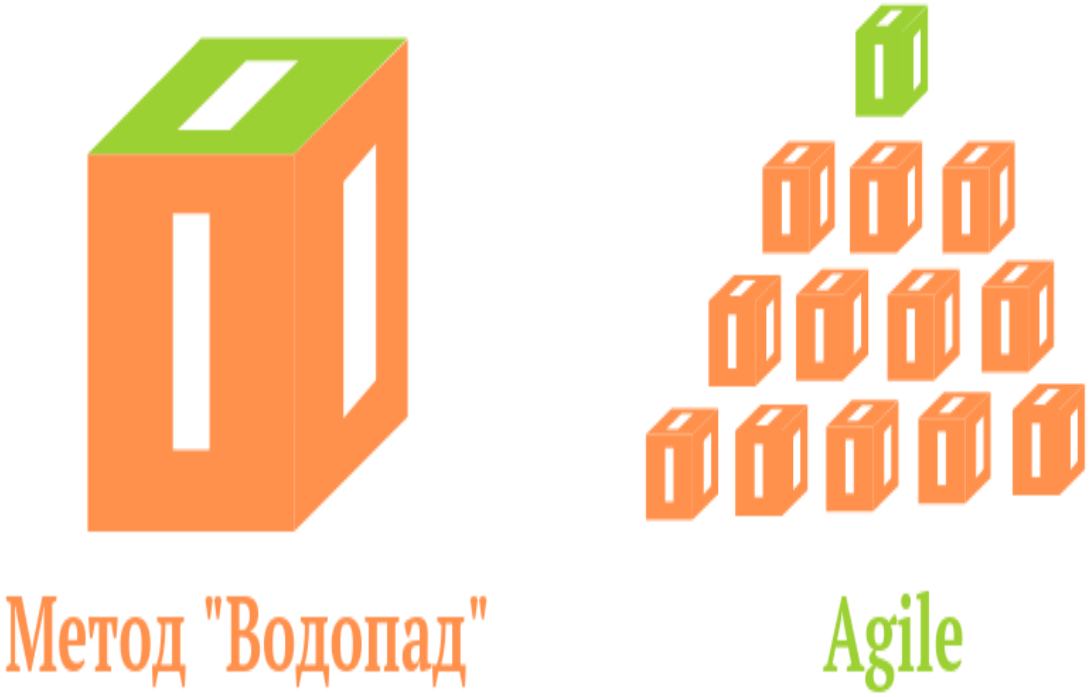

Рисунок 1 – Різновид методів

1.1.2 Agile Manifesto

Agile Manifesto — це виклад основних цінностей та принципів розробки програмного забезпечення. Провідний маніфест з розробки програмного забезпечення був створений в 2001 році, і це декларація чотирьох важливих правил та дванадцять принципів, які слугують керівництвом для людей, що займаються спритною розробкою програмного забезпечення. Його створили сімнадцять професіоналів, які вже практикували спритні методи, такі як XP, DSDM, SCRUM, FDD тощо, зібрані в засніжених горах американського штату Юта, скликані Кентом Беком.

4 Основні цінності Agile Manifesto:

– Особи та взаємодія між процесами та інструментами — перше значення наголошує на колективній роботі та спілкуванні. Ми повинні

розуміти, що розробка програмного забезпечення є людською діяльністю і що якість взаємодії між людьми є життєво важливою. Інструменти є важливою частиною розробки програмного забезпечення, але створення чудового програмного забезпечення набагато більше залежить від колективної роботи, незалежно від інструментів, які команда може використовувати;

- Робота програмного забезпечення над вичерпною документацією — Документація має своє місце і може бути чудовим ресурсом або довідковою інформацією для користувачів та колег. Однак основною метою розробки програмного забезпечення є розробка програмного забезпечення, яке пропонує бізнес переваги, а не велику документацію;
- Співпраця з клієнтами під час узгодження контракту — команди розробників повинні тісно співпрацювати та часто спілкуватися зі своїми клієнтами. Слухаючи та отримуючи відгуки команди зрозуміють, чого насправді хочуть усі зацікавлені сторони;
- Реагування на зміни після дотримання плану — зміни є реальністю у розробці програмного забезпечення, реальністю, яку повинен відображати ваш процес Програмного забезпечення. План проекту повинен бути достатньо гнучким, щоб змінюватися, як того вимагає ситуація.

Вперше система управління канбан була створена фірмою «Toyota». У 1959 році ця фірма почала експерименти з системою канбан, і в 1962 році запустила процес перекладу всього виробництва на цей принцип.

Поява терміна канбан пов'язано з перерахуванням стандартних операцій: майстри дільниць перераховували виконувані роботи на папері і вивішували їх на видному місці поруч з такими ж списками майстрів інших ділянок. Це була публічна, загальнодоступна дошка, на яку прикріплювали кольорові стікери (картки) з завданнями.

У міру проходження завдань від одного етапу до іншого їх переносили в відповідний стовпець. В даний час система реалізована у вигляді онлайн дощок і широко використовується в різних галузях промисловості, виробництва, торгівлі та фінансів.

Канбан дошка дозволяє візуалізувати робочий процес, своєчасно виявляти проблеми в процесі роботи, здійснювати контроль і самоконтроль, вести облік виконаних завдань, поліпшити взаємодію в колективі, вести облік клієнтів і т. П.

Приклад найпростішої дошки включає стовпці: «Зробити», «В роботі» і «Зроблено». У стовпчиках наклеєні стікери (картки) з завданнями. Раніше люди виконували меншу кількість завдань. Життя вимагає зростання обсягів завдань, що виконуються кожним підрозділом. Зі збільшенням кількості завдань їх стало складніше запам'ятовувати, і ми почали записувати їх в щоденники. Потім в файл Excel. Потім з'явилися електронні органайзери, які теж полегшували частина роботи із запам'ятовування.

Раніше ми запам'ятовували номери телефонів один одного. З появою програм, в телефонах ця звичка пішла в минуле за непотрібністю. З появою канбан дощок, аналогічним чином відпала необхідність запам'ятовувати завдання. Якщо взяти для прикладу групу «продажників» в кількості 5-9 чоловік, то приблизну кількість завдань у кожного може доходити до 20-30 штук. Кожен день 10-15% завдань змінюється, так як надходять нові й нові. У деяких завданнях змінюються терміни, деякі дробляться на додаткові підзадачі, таким чином виникають нові завдання. Відповідно змінюються терміни дати, суми, назви клієнтів і контрагентів, та й самі завдання.

Фізично тримати менеджеру це в голові, без постійного повторення — неможливо, та й потреби в цьому немає. А як запам'ятати подробиці всіх завдань їх керівнику, коли кількість завдань в роботі йде далеко за сотню?

Кажуть, що є люди, які в змозі абсолютно всі завдання запам'ятовувати постійно, працюючи в режимі багатозадачності. Клінічні випадки ми розглядати не

будемо, а в більшості випадків співробітник, який стверджує, що він все пам'ятає, помиляється або лукавить.

На практиці віра в такі твердження призводить до того, що до 30% завдань просто забувається в потоці інформації, що призводить до втрат (недоотриманого прибутку), необгрунтованим авралів на шкоду іншим задачам.

Час, раніше необхідне для заучування назв компаній, імен, дат і сум, які регулярно змінюються, тепер можна використовувати більш продуктивно, на виконання поставлених завдань.

Розташування стовпців на дошці і рух по ній стікерів (карток) зазвичай становить найбільшу складність при створенні своєї першої канбан дошки. На канбан дошці стовпчики розподіляються по етапах проходження завдань як на рисунку 2.

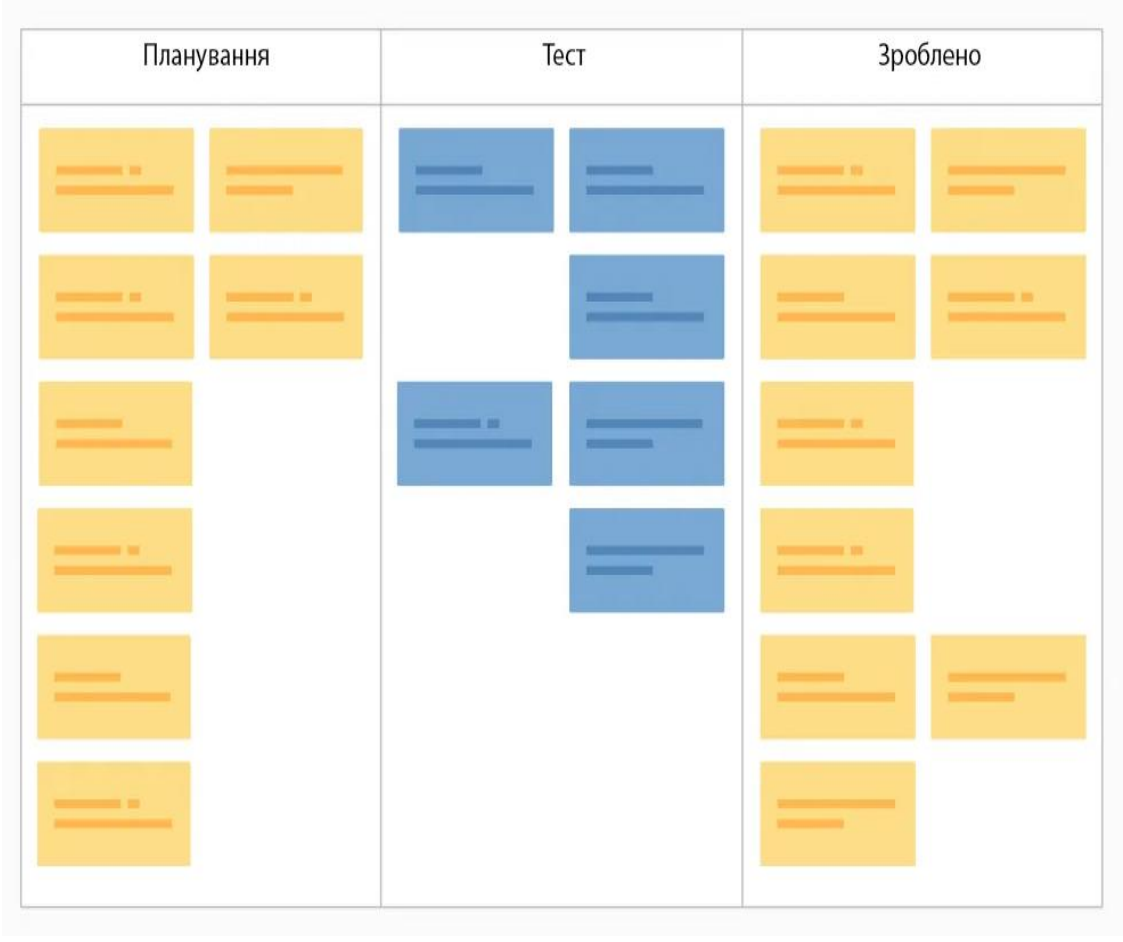

Рисунок 2 – Канбан дошка

Найпростіший варіант, визначитися з тим, як саме краще структурувати дошку, це взяти аркуш ватману і приклеїти його на стіну. Розкреслити його на стовпці і перерахувати на ньому всі основні операції, що виконуються підрозділом зараз. Потім попросити кожного співробітника написати на стікері завдання, якими він займається в даний час і приклеїти їх у відповідний стовпець. У міру просування завдань по етапах їх переносять (переклеюють) з шпальти в стовпець до тих пір, поки стікер з завданням не потрапить в стовпець «Виконано».

Таким чином, протягом декількох днів, колектив отримає уявлення про роботу з канбан дошкою і можна буде переносити отриману структуру в електронний вигляд.

Те ж саме відбувається і на електронній канбан дошці. Ви чіпляєте мишкою стікер з завданням і перетаскуєте його з шпальти в стовпець у міру просування завдання. Можна також підняти або опустити будь-стікер з завданням, що буде сигналом для співробітника, зайнятися цим завданням, відповідно раніше або пізніше.

Загальний принцип: чим вище на дошці розташований стікер з завданням, тим завдання більш пріоритетна для менеджера, на сьогоднішній день.

1.2 Ціль та задача додатку

Основне завданням проекту — є створення програми, яка полегшить процес планування нового проекту. А також поглибити та узагальнити теоретичні знання, для обраного нами напрямку дослідження. Планування проекту — це безперервний процес визначення найкращого напрямку дій для досягнення цілей проекту з урахуванням поточної ситуації.

Планування — це найважливіший процес управління проектами, який визначає всю проектну діяльність у часі. Процеси планування відбуваються протягом життєвого циклу проекту, починаючи від попереднього зведеного плану як частини концепції проекту і закінчуючи детальним планом роботи на заключній

фазі проекту. Всю цю роботу виконує PM (керівник проекту). У той же час плани вдосконалюються та деталізуються в міру прогресу проекту.

Етап планування визначає організацію, методи та засоби управління реалізацією проекту як системи в цілому, так і в контексті окремих її етапів та елементів.

Планування логічно пов’язане з іншими важливими етапами процесу управління, такими як ініціювання, організація та контроль впровадження, аналіз та регулювання та закриття проекту. Метою планування — є побудова моделі для реалізації проекту.

Основним результатом фази планування — є зведений план реалізації проекту, який узагальнює результати планування за всіма функціями управління проектами. Цей документ є головним і вирішальним при реалізації проекту, він служить зразком (планом дій) та прогнозом стану проекту та його середовища. У процесі реалізації проекту можуть відбуватися зміни як усередині проекту, так і за його межами. Тому основною метою планування є безперервна підтримка прогресу проекту на шляху його успішного завершення.

Об'єктами планування в проекті є:

- Предметна область;
- Час;
- Вартість;
- Якість;
- Організація;
- Комунікації;
- Ризики;
- Поставки і контракти;
- Зміни;
- Інші компоненти проекту;
- Інтеграційний план.

По-друге це планування предметної області проекту. Предметна область проекту (Project Scope) — сукупність продуктів і послуг, виробництво яких повинно бути забезпечено в результаті завершення здійснюваного проекту. Предметну область проекту визначають цілі, результати та роботи проекту. В процесі життя проекту, всі складові предметної області проекту можуть у змінах.

Цілі результати, роботи і їх характеристики можуть змінюватися або уточнюватися як в процесі розробки проекту, так і в міру досягнення проміжних результатів.

Планування предметної області проекту включає наступні завдання і процедури:

- Аналіз поточного стану та уточнення цілей і результатів проекту;
- Уточнення основних характеристик проекту;
- Підтвердження і уточнення критеріїв успіху і невдач проекту;
- Аналіз і коригування обмежень і припущень, прийнятих на стадії ініціації проекту;
- Вибір критеріїв оцінки проміжних і остаточних результатів створення проекту;
- Побудова структурної декомпозиції предметної області проекту.

Злагоджена праця завжди залежить від планування часу проекту. Злагоджена робота всіх учасників проекту організовується на основі календарних планів або розкладів робіт проекту, основними параметрами яких є: терміни виконання, ключові дати, тривалості робіт та інше. Календарними планами називають проектно-технологічні документи, встановлюють повний перелік робіт проекту, їх взаємозв'язок, послідовність і терміни виконання, тривалості, а також виконавців і ресурси, необхідні для виконання робіт проекту. Планування проекту по часових параметрів полягає в складанні різних календарних планів (розкладів робіт), які відповідають всім вимогам і обмеженням проекту і його частин. Календарні плани

планування проекту складаються на весь життєвий цикл проекту і його етапи, для різних рівнів управління і учасників проекту.

1.3 Вибір архітектури проекту і інструментів розробки

C# — це мова програмування, розроблена та запущена корпорацією Майкрософт у 2001 році. C# — це проста, сучасна та об'єктно-орієнтована мова, яка забезпечує сучасним розробникам гнучкість та можливості для побудови програмного забезпечення, яке не тільки буде працювати сьогодні, але й буде застосовуватися протягом багатьох років у майбутнє.

Основні характеристики мови C # включають:

- Сучасна і легка;
- Швидкий і відкритий код;
- Крос-платформена;
- Безпечна;
- Універсальниа;
- Еволюціонує.

C# — це проста, сучасна та об'єктно-орієнтована мова програмування. Метою C# була розробка мови програмування, яку не тільки легко вивчити, але й підтримувати сучасні функціональні можливості для всіх видів розробки програмного забезпечення.

Якщо розглянути історію мов програмування та їх особливості, кожна мова програмування була розроблена з певною метою для вирішення конкретної потреби на той момент.

Однак мова C# була розроблена з урахуванням потреб бізнесу та підприємств. Мова C# була розроблена для бізнесу для створення всіх видів програмного забезпечення за допомогою однієї мови програмування.

C# надає функціонал для підтримки сучасної розробки програмного забезпечення. C# підтримує потреби в інтернеті, мобільному зв'язку та розробці додатків. Деякі з сучасних функцій мови програмування, які підтримує C#, — це загальні типи, типи змінних, автоматична ініціалізація типів та колекцій, лямбда-вирази, динамічне програмування, асинхронне програмування, кортежі, узгодження шаблонів, розширена налагодження та обробка винятків тощо.

На синтаксис мови C# впливають C++, Java, Pascal та деякі інші мови, які легко прийняти. C# також уникає складності та неструктурованих мовних особливостей.

C# має відкритий вихідний код у .NET Foundation, який керується та працює незалежно від Microsoft. Специфікації мови C#, компілятори та відповідні інструменти — це проекти з відкритим кодом на Github. Хоча розробкою функцій мови C# керує Microsoft, спільнота з відкритим кодом дуже активно займається розробкою та вдосконаленням мови.

C# — це швидке порівняння з кількома іншими мовами програмування високого рівня. C # 8 має багато поліпшень продуктивності.

Ви можете створювати програми .NET, які можна розгорнути на платформах Windows, Linux та Mac. Програми C# також можна розгорнути у хмарі та контейнерах.

C# не дозволяє перетворювати типи, які можуть призвести до втрати даних або інших проблем. C# дозволяє розробникам писати безпечний код. C# також зосереджується на написанні ефективного коду.

Ця мова дуже надійна як швейцарський армійський ніж. Хоча більшість мов програмування були розроблені з певною метою, C# був розроблений для щоб зробити C#. Ми можемо використовувати C# для створення сучасних сучасних програмних додатків. C# може використовуватися для розробки всіх видів додатків, включаючи клієнтські програми Windows, компоненти та бібліотеки, служби та API, веб-програми, мобільні програми, хмарні програми та відеоігри.

Ось перелік типів програм, які C# може створювати:

- Клієнтські програми Windows;
- Бібліотеки та компоненти Windows;
- Служби Windows;
- Веб-додатки;
- Веб-сервіси та веб-API;
- Власні мобільні додатки для iOS та Android;
- Послуги бекенда;
- Хмарні програми та служби Azure;
- Базова база даних за допомогою інструментів ML / Data;
- Програмне забезпечення для сумісності, таке як Office, SharePoint, SQL Server тощо;
- Штучний інтелект та машинне навчання;
- Технологія блокчейнів та розподілена книга, включаючи криптовалюту;
- Пристрої інтернет речей (IoT);
- Ігрові приставки та ігрові системи;
- Відео ігри.

Середа CLR попутно виконує ще багато побічних функцій:

- Видалення «програмного» сміття;
- Робота з винятками;
- Розподіл ресурсів;
- Контроль типізації;
- Управління версіями;
- Типізація;
- Управління версіями.

В результаті код C# вважається керованим, тобто він компілюється в двійковий вид на призначеному для користувача пристрої з урахуванням особливостей встановленої системи.

1.4 Аналіз ринку. Актуальність дипломної роботи

На ринку є багато різних програм створенних за для продуктивної роботи команди програмістів. Компанії Jira та Trello вже давно є лідерами у цілому світі з створення програм для планування проектів. Але водночас ці програми є дуже складними у використанні для звичайного користувача. На рисунку 3 ми можемо побачити інтерфейс програми Trello.

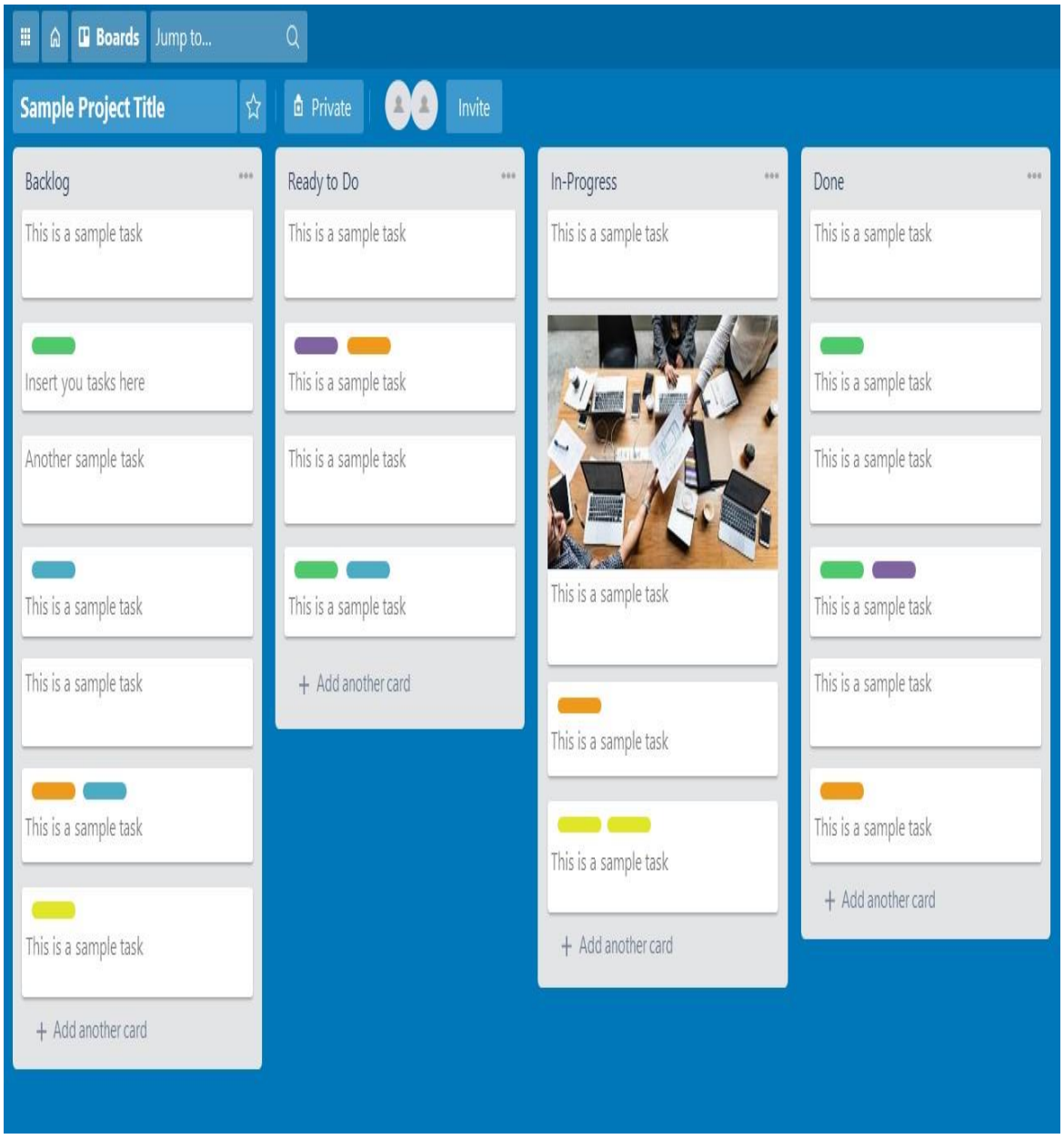

Рисунок 3 – Інтерфейс програми Trello

Мінуси роботи з програмою Trello:

- Чим більше проектів веде одна людина тим нерозбірливою становится вікно програми;
- Неможливість користування оффлайн;
- Программа не може відслідковати час;
- Немає можливості редагувати або змінювати опис до проекту;
- Неможливість змінювати підзадачі.

Jira — комерційна система для відслідковування помилок, та робочого процесу команди розробників.

В ній передбачені робота над проблемами та часом в роботі над проектом. У Jira є велика кількість додатків які синхронизуються між собою, що надає великий функціонал робочої програми. Інтерфейс Jira можна підлаштовувати під себе кожен користувач, як на рисунку 4.

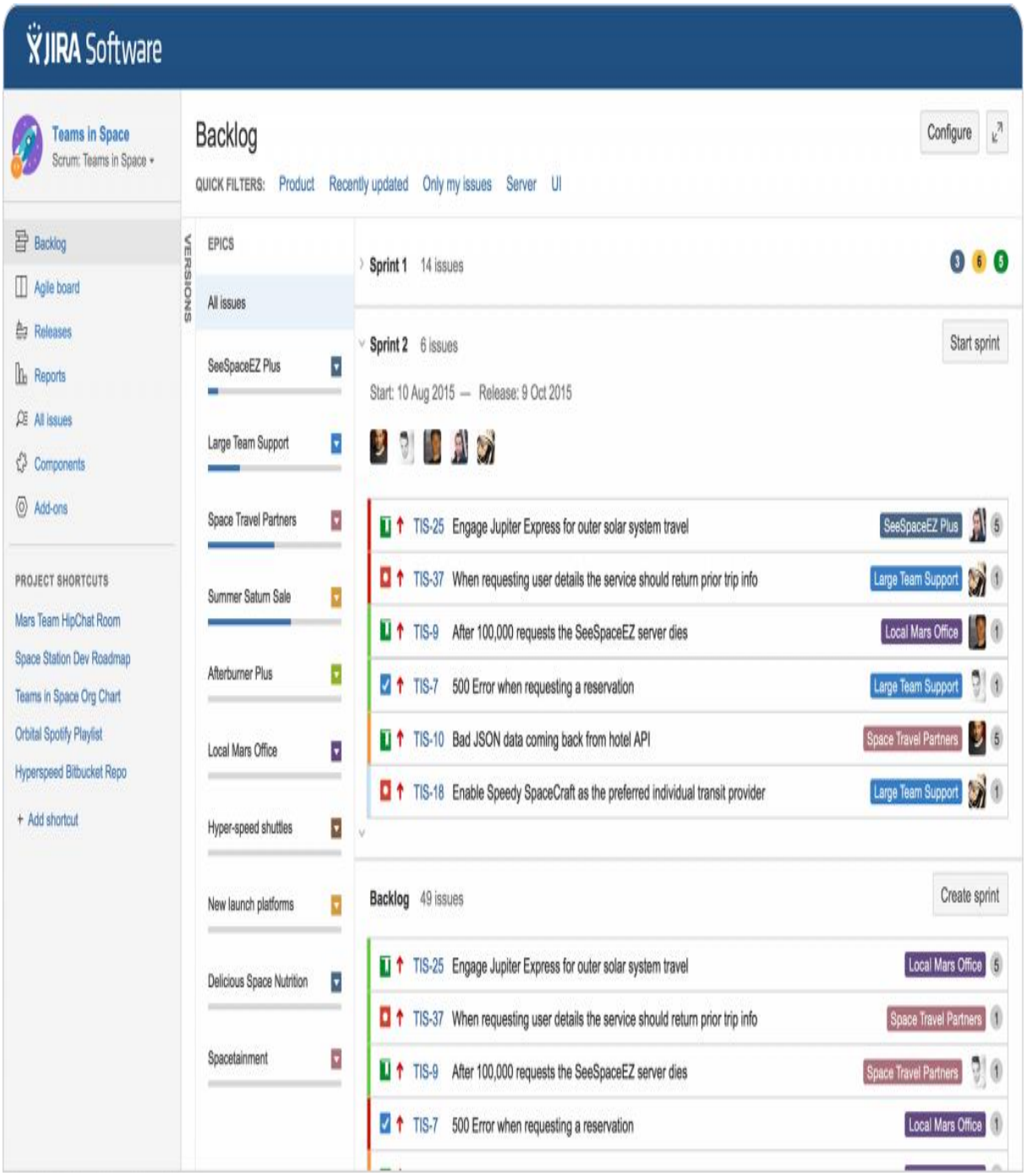

Рисунок 4 – Інтерфейс програми Jira

Мінуси роботи з програмою Jira:

- При великій кількості підзадач процес виконування головної буде дуже складно відстежувати;
- Досить складна в освоєнні.

Впершу чергу програма, звичайно ж, повинна вирішувати поставлені завдання і добре виконувати свої функції, причому в різних умовах.

Вимога, щоб архітектура системи володіла гнучкістю і розширюваністю (тобто була здатна до змін і еволюції) є настільки важливим, що воно навіть сформульовано у вигляді окремого принципу — «Принципу відкритості/ закритості» (Open-Closed Principle — другий з п'яти принципів SOLID): програмні суті (класи, модулі, функції і т.п.) повинні бути відкритими для розширення, але закритими для модифікації.

Не варто відразу рубати додаток на сотні класів. Декомпозицію треба проводити ієрархічно — спочатку систему розбивають на великі функціональні модулі, підсистеми, що описують її роботу в найзагальнішому вигляді. Потім, отримані модулі, аналізуються більш детально і, в свою чергу, діляться на під-модулі або на об'єкти.

Перед тим як виділяти об'єкти розділіть систему на основні смислові блоки, для невеликих додатків двох рівнів ієрархії часто виявляється цілком достатньо — система спочатку ділиться на підсистеми — пакети, а пакети діляться на класи.

Ця думка, при всій своїй очевидності, не так банальна як здається. Наприклад, в чому полягає суть такого поширеного «архітектурного шаблону» як модель-вид-контролер (MVC)? В проектуванні додатків суть полягає в тому, що будь-який користувальницький додаток спочатку ділиться на два модуля — один з яких відповідає за реалізацію власне самої бізнес логіки (модель), а другий — за взаємодію з користувачем (для користувача інтерфейс або подання). Потім, для того щоб ці модулі могли розроблятися незалежно, зв'язок між ними послаблюється за допомогою паттерна «Спостерігач» (докладно про способи ослаблення зв'язків буде розказано далі) і ми фактично отримуємо один з найпотужніших і затребуваних «шаблонів», які використовуються в даний час.

2 ПРОЕКТНА ЧАСТИНА ДОДАТКУ

2.1 Аналіз вимог до розроблюваної системі

2.1.1 Основні вимоги до ПЗ

Вимоги до програмного забезпечення — це детальне зображення використовуваного фреймворку. Вимоги можуть поширюватися від унікальних статей унікальних формулювань адміністрацій або обмежень до точкової математичної функціональної специфікації. Обговорюється важливість критеріїв якості у встановленні вимог, подібно до впливу підтвердження якості серед вимог на різні частини вдосконалення. Відмінні підходи до якості класифікуються як спільні, або обговорюються щодо їх впливу на вимоги. З огляду на методики, пояснюються майбутні труднощі.

Системні вимоги складаються з наступних умов, які повинні бути виконанні у процесі розробки ПЗ:

- склад виконуваних функцій системи (запуск, швидкість реакції й ін.);
- конфіденційність, безпека і захист даних;
- відмовостійкість;
- одночасність доступу користувачів до системи;
- час очікування відповіді при звертанні до системи (продуктивність);
- положення стандартів з виконання сформульованих вимог.

Зрозуміло, що якість SRS є найбільшим фактором процвітання та розчарування прогресивного підприємства. Найважливішим моментом є створення роботи із забезпечення якості програмного забезпечення як життєво важливої частини SDLC та забезпечення загальної якості переданого програмного забезпечення найбільш вражаючою потребою для всіх, хто бере участь у проекті. На думку експертів, стадія вимог є основою програмного забезпечення. Вони складають основу для етапів розробки програмного забезпечення. Необхідно, щоб

повний набір вимог ініціалізувався на початковій стадії процесу розробки програмного забезпечення. Метою дослідження вимог є визнання та вираження вимог, що визначають вимоги та цілі користувачів. Вимоги до програмного забезпечення не можуть бути визнані виключно шляхом перевірки їх впливу на фактори нижчого рівня. Тому визначення вимог - це послідовна процедура, яка працює зверху вниз і знизу вгору. Після того, як розроблено набір вимог до програмного забезпечення верхнього рівня, важливо розподілити та перевести їх на нижчий рівень. Важливо, щоб і надалі було використано більше факторів, щоб забезпечити виконання всіх вимог до програмного забезпечення на стадії проектування.

2.1.2 Виявлення вимог

Існує кілька типів вимог, які ми можемо побачити, коли робимо проект. Ці вимоги можна класифікувати наступним чином, це ті вимоги, що стосуються технічного управління, а також вимоги замовника. Існують певні твердження про факти та припущення, які визначають ці очікування системи. Ці твердження, як правило, пов'язані з цілями місії, середовищем та обмеженнями системи. Вони додатково містять показники ефективності точної системи та наскільки вона відповідна. Функціональні вимоги — це компонент програмного забезпечення. З цією вимогою ми можемо розписати як вхідні дані, результати та поведінку. Що слід виконати, які завдання необхідно визначити, пояснюються у функціональних вимогах. На додаток до використання функцій верхнього рівня в функції аналізу також можливі.

Вимоги до продуктивності системи є ступенем виконання місії або функції. Все це загалом вимірюється з точки зору кількості, якісного охоплення, своєчасності чи готовності. При аналізі вимог, виходячи з факторів, що регулюють життєвий цикл системи, вимоги до продуктивності розроблятимуться інтерактивно для всіх визначених функцій; вони також будуть характеризуватися залежно від ступеня визначеності в оцінці, того, наскільки вони є критично важливими для

успіху системи та того, як вони пов'язані з іншими вимогами. Вимога до продуктивності означає швидкість введення, передачі та обробки даних. Як приклад в системі торгових точок (POS) введення даних повинно бути швидким, але не обробкою даних. Тож виконання цих аспектів повинно відповідати вимогам замовника, бізнесу та робочого середовища.

2.1.3 Принципи планування

Будь-яка робота займе рівно стільки часу, скільки на неї відведено або більше. (Закон Паркінсона). Люди відкладають всяку роботу на останній момент часу. (Синдром студента). Працівники схильні давати оптимістичні прогнози щодо трудомісткості невеликих робіт. Виключаючи випадки коли їх позбавляли премії за порушення оцінок, в цих випадках оцінка песимістична і завищена. Працівники дають песимістичні прогнози щодо трудомісткості великих робіт. Тому що хочуть встигнути «напевно» і провести «звитяжну війну». Люди ніколи не починають роботу саме в той час, коли робота запланована і ніколи не закінчують виконання завдання вчасно. Всі проекти пов'язані з невизначеністю (Закон Мерфі — якщо щось може піти не так, саме це і станеться. Ми ніколи не знаємо, коли закон Мерфі себе проявить).

Реальність ніколи не буде відповідати плану. Як би добре ми не планували наш проект, він не буде охоплювати все. Обсяг роботи за проектом не постійний. Суть проекту — у створенні чогось унікального, і ми ніколи не знаємо, що саме потрібно робити, поки не почнемо виконувати роботу. Програмісти ніколи не роблять рівно те, що написано в технічному завданні. Вони роблять більше або менше або іншим способом. Вони можуть ефективно виконувати тільки одну задачу в один момент часу.

Виходячи з принципів планування і зовнішніх зобов'язань організації необхідно застосовувати два рівня планування:

- Перший рівень — стратегічне та великоблочне планування проектів на основі правил їх виконання;

– Другий рівень — тактичне планування спринтів і визначення коротких цілей.

Така розбивка пов'язана з принципом "необхідності і достатності", невизначеністю змісту проекту і непередбачуваності поведінки людей.

Рівень планування проектів забезпечує впорядкування почерговості виконання робіт організацією, задає зрозумілі цілі, які фокусують мету і забезпечують контроль виконання зобов'язань.

Рівень тактичного планування спринтів забезпечує короткі і зрозумілі цілі, для співробітників які реалізують проект і враховує високу невизначеність способу реалізації.

Планування, як процес, не повинен обмежуватися сесіями в форматі "зробив і забув". Це процес, що вимагає стільки ж зусиль, як і проектування продукту. Планування і вибір пріоритетів майбутніх робіт повинно відбуватися в той час, поки поточні роботи ще не завершені.

Планування майбутніх проектів і розвитку продуктів виконується регулярно. Сесії планування "вшиті" в робочий процес. У плануванні може взяти участь будь-який співробітник, якщо бачить у ньому свою користь. Наради з планування збираються регулярно для синхронізації всіх планів і бачення рішень. Розробка проектів складається з початкової реалізації планів його виконання.

Коли будується розклад проектів, проводиться збори для прийняття рішень в організації на тривалий період. Маючи календарний розклад проекту і черговість виконання завдань, не потрібно витрачати зусилля на вибір найважливішого і пріоритетного завдання кожного разу, коли це необхідно. Це економить зусилля на прийняття рішень.

Розклад проекту як правило визначає пріоритет завдань, якого слід дотримуватися. Якщо щось змінилося у зовнішнім чи внутрішнім середовищі, то слід виконувати перепланування для обліку всіх зацікавлених сторін і зобов'язань організації. Продовження роботи за старим планом — небезпечно для організації.

Наявність календарного розкладу дозволяє краще розуміти приблизні терміни досягнення цілей, і на основі цього синхронізувати інші роботи.

2.2 Опис функцій та алгоритму системи

Функції додатку зображені на рисунку 5 у вігляді UML діаграми прецендентів з одним актором.

На діаграмі зображено, що може робити актор. Гравець може:

- Створювати нову дошку
- Використовувати вже існуючі дошки
- Додавати нові завдання
- Переміщувати завдання між колонками
- Зберігати дошку

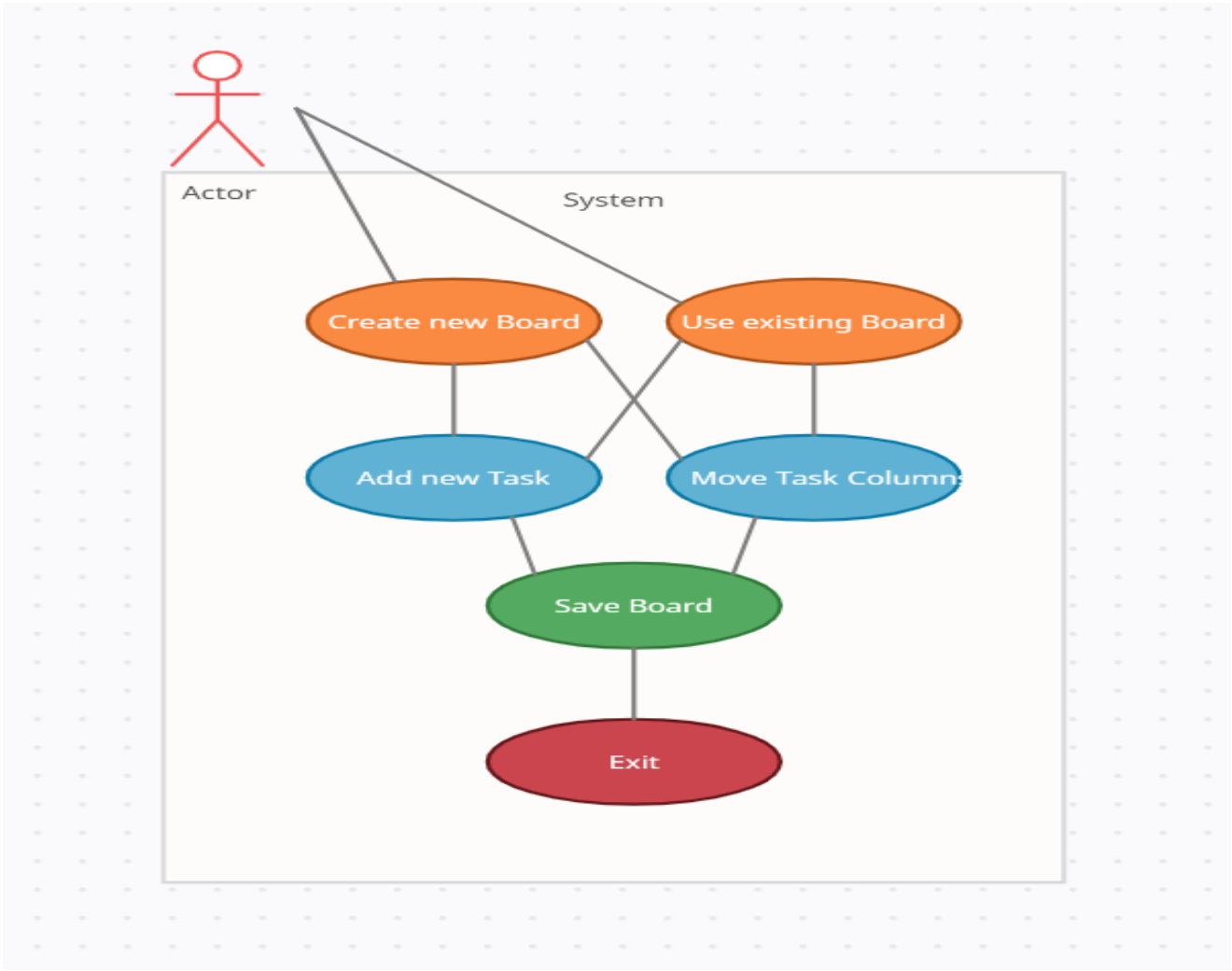

Рисунок 5 – UML діаграма прецендентів

В процесі написання програми було розроблено ієрархію класів представлену на рисунку 6.

Рисунок 6 – Ієрархія класів

На рисунку 7 зображено циклічний, розгалужений алгоритм роботи програми, набір інструкцій, які описують порядок дій виконавця

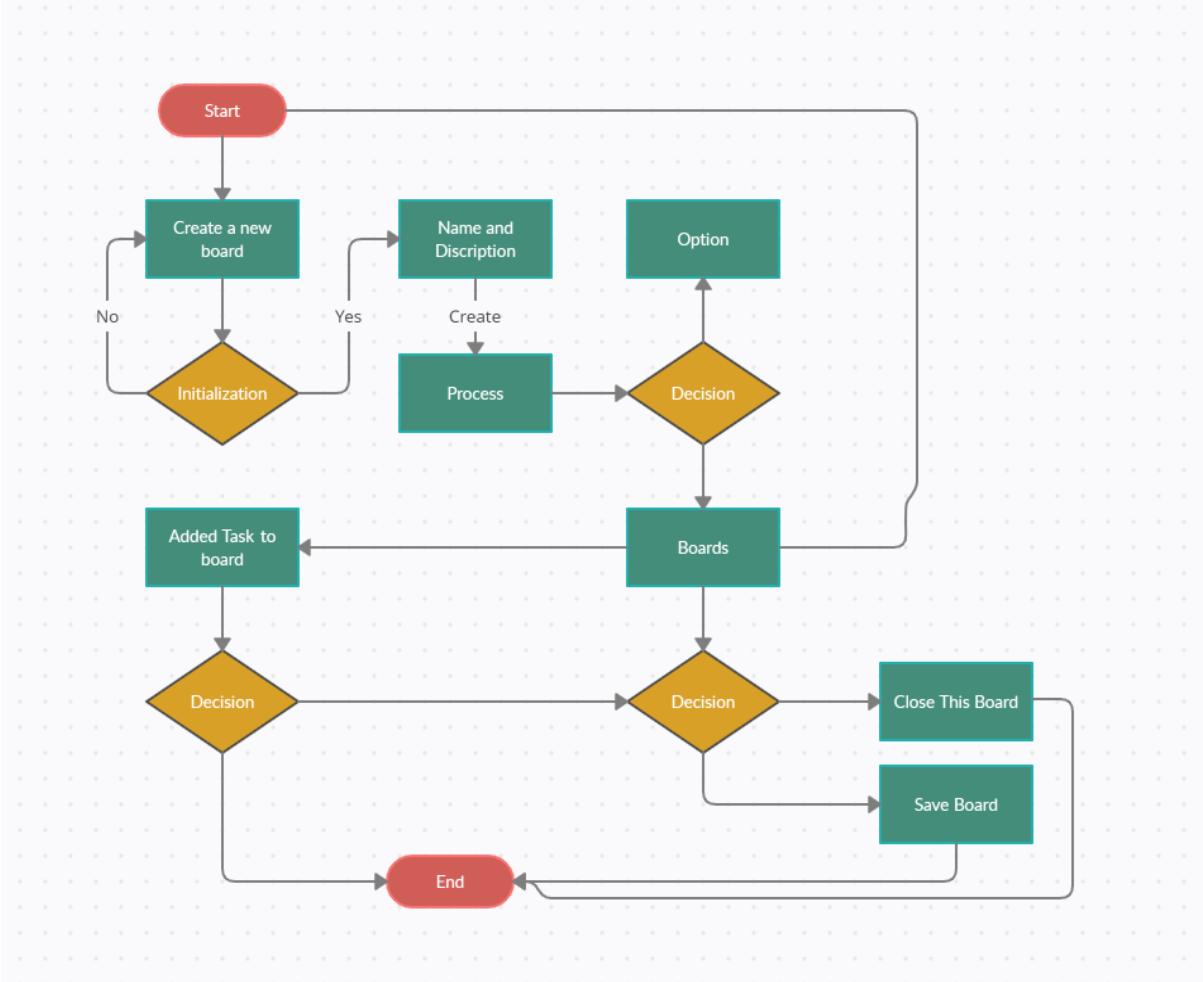

Рисунок 7 – Алгоритм роботи програми

2.3 Вимоги до апаратного та програмного забезпечення

Додаток був створений за для того щоб він міг використовуватися на будь-якому комп'ютері. Тому апаратне та програмне забезпечення для запуску програми є мінімальним.

- Процесор – Intel Pentium x64, не нижче 300 MHz
- Обсяг ОЗУ – не нижче 64 Мб
- Відеокарта: NVIDIA Geforce GT 630M
- Вільний простір на диску — 100 Мб
- Монітор з роздільною здатністю не менше 800x600, TrueColor
- Клавіатура – Windows-сумісна

- Наявність комп’ютерної миші або touch screen
- Операційна система — Windows / XP / 7 / 8 / 8.1 / 10

2.4 Дизайн

Було розробленно з нуля дизайн логотипа програми, за усіма виробничими стандартами рисунок 8. Кінцевий варіант був створенний в Photoshop рисунок 9.

Рисунок 8 – Розробка логотипу в Paint

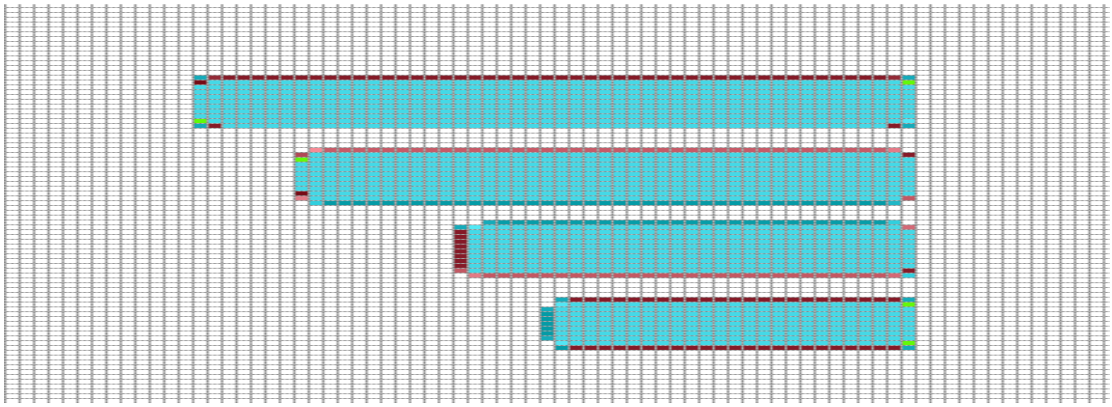

Рисунок 9 – Розробка логотипу в Photoshop

Спільні елементи керування автоматично відображають системну тему та колір акценту, працюють з усіма типами вводу та масштабуються на всіх

пристроях. Программа є повністю адаптивною до кожної операційної системи Windows, так як була розроблена за допомогою технології WPF рисунок 10.

Рисунок 10 – Головний екран

На головному екрані ви можете побачити назву програми та її логотип. Тут користувач починає ознайомлення з додатком і може розпочати роботу, натиснувши кнопку посередині екрану, щоб створити нову дошку.

Якщо користувач вже працював з додатком, то він може вибрати вже існуючий проект, натиснувши на нього зліва на екрані програми як на рисунку 11.

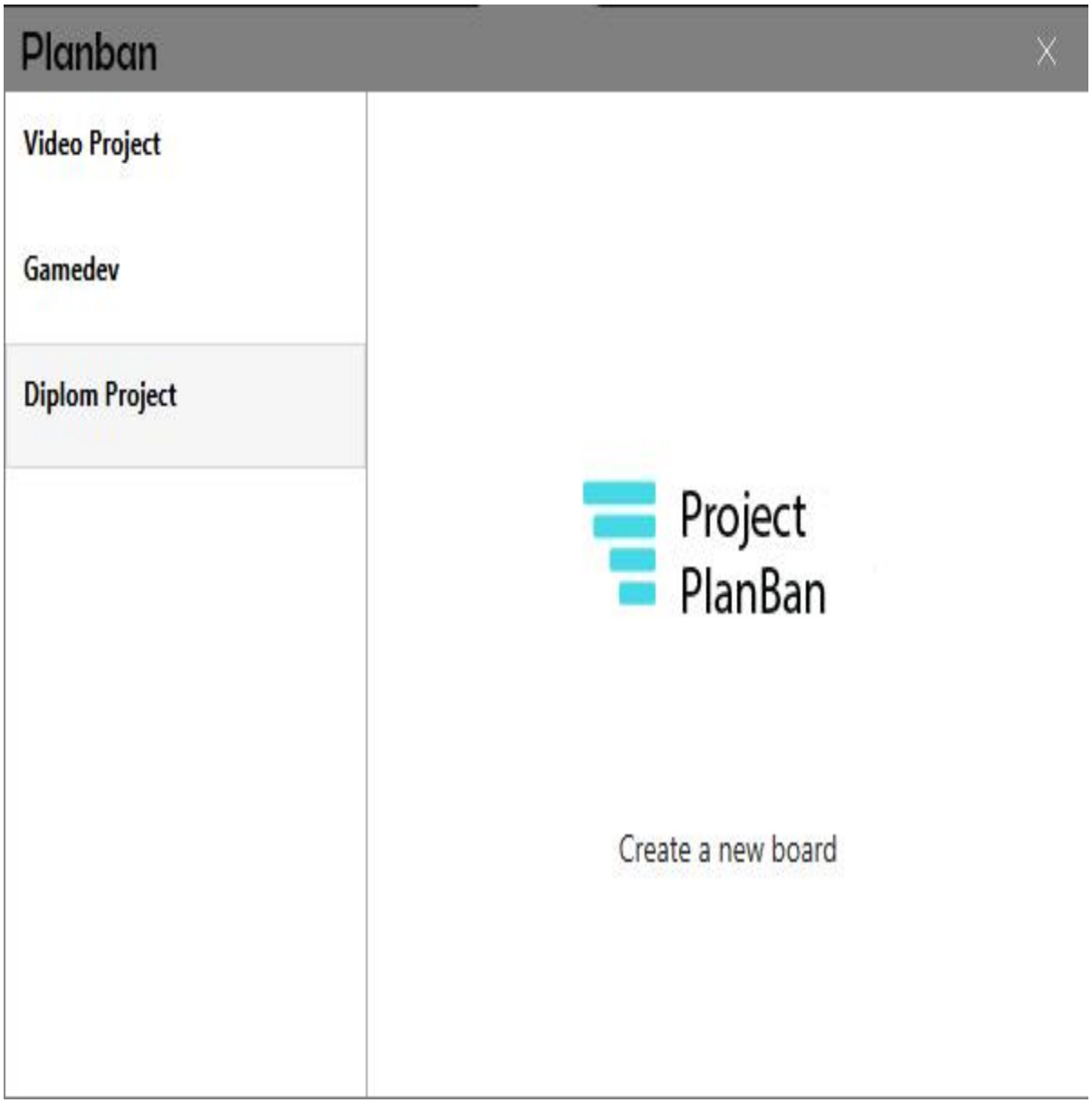

Рисунок 11 – Вибір існуючого проекту

Після того як користувач натиснув кнопку створити нову дошку, з'являється вікно з назвою задачі та її змістом, в якій користувач детально описує принцип виконання який показано на рисунку 12.

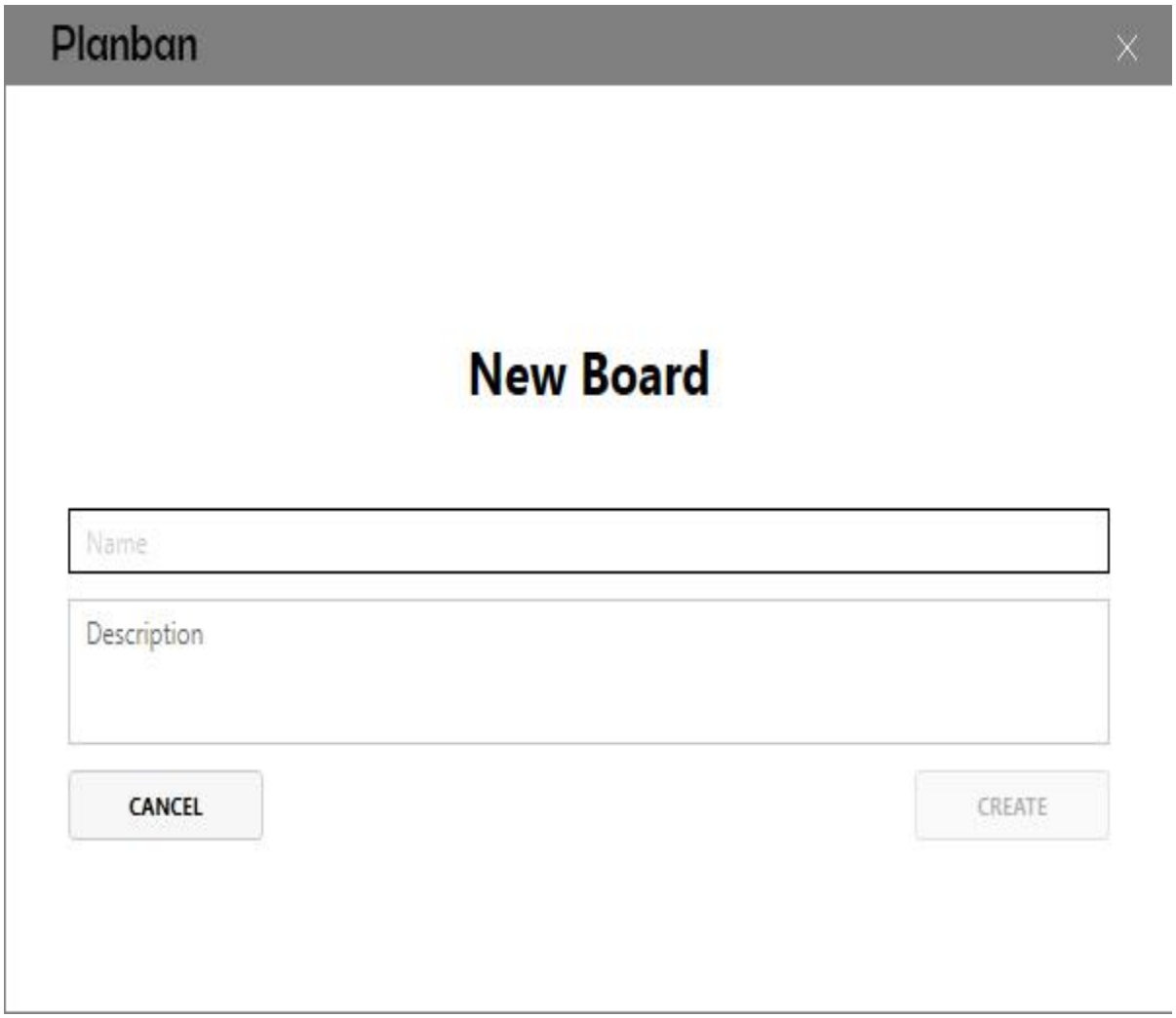

Рисунок 12 – Створеня назви та опис додатку

В головному вікні можна додати подію, зробити їй назву та надати відповідний пріорітет . Можна відстежувати дату створення або змінення тієї чи іншої задачі (рисунок 13).

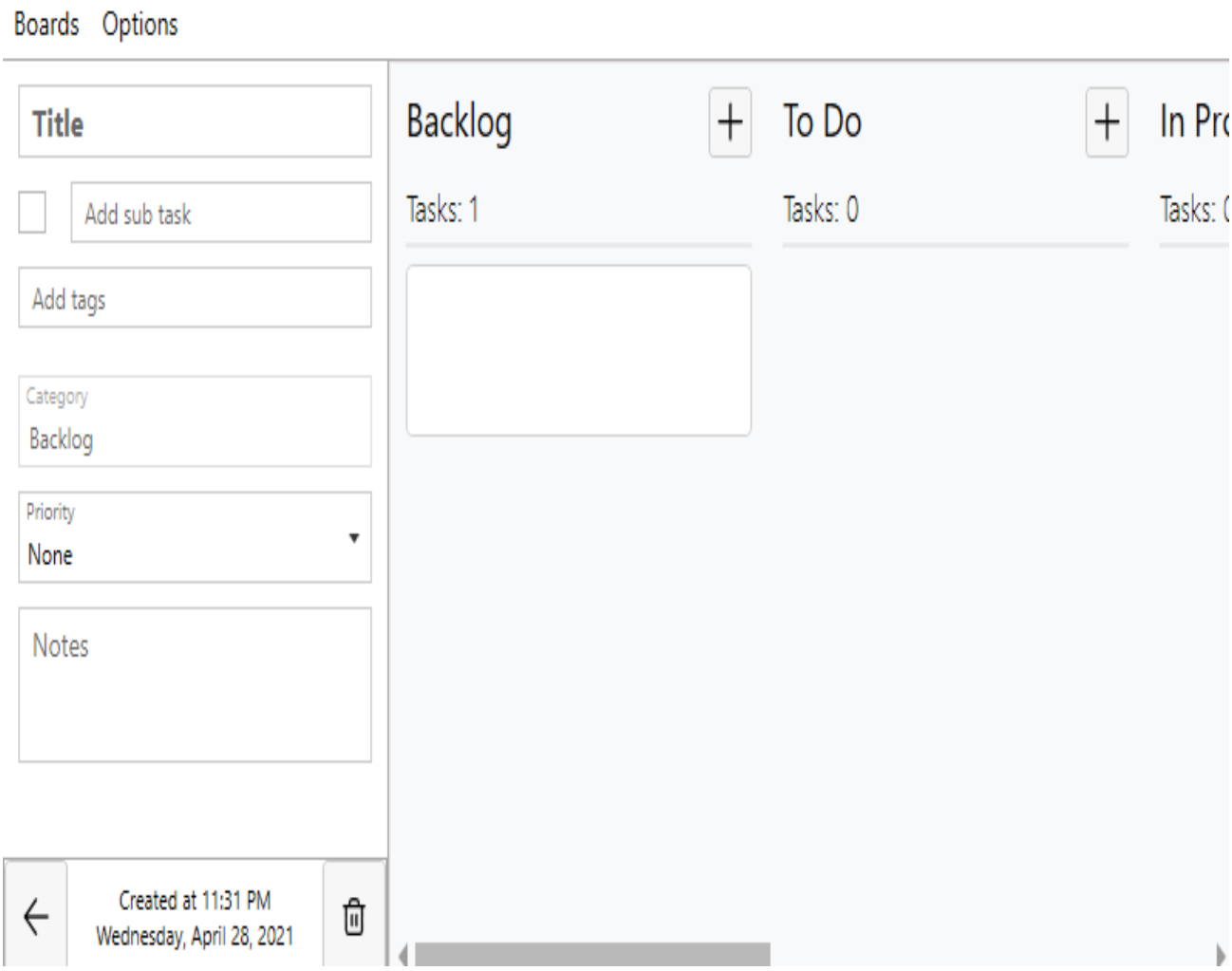

Рисунок 13 – Створення задачі в головному меню

В головному меню можна створювати назву задачі, яка буде виділена жирним шрифтом. Також створення підзадач які можна буде відслідковувати у маленькому віконечку під головною назвою задачі. У кожної задачі є свій пріорітет для виконання. Пріорітет у події може бути різних кольорів (рисунок 14). Зелений колір малий пріорітет, жовтий зазначає середній, а червоний це задача з найвищим пріорітетом для виконання. Так команда розробників буде розуміти які задачі треба робити швидше для здачі проекту.

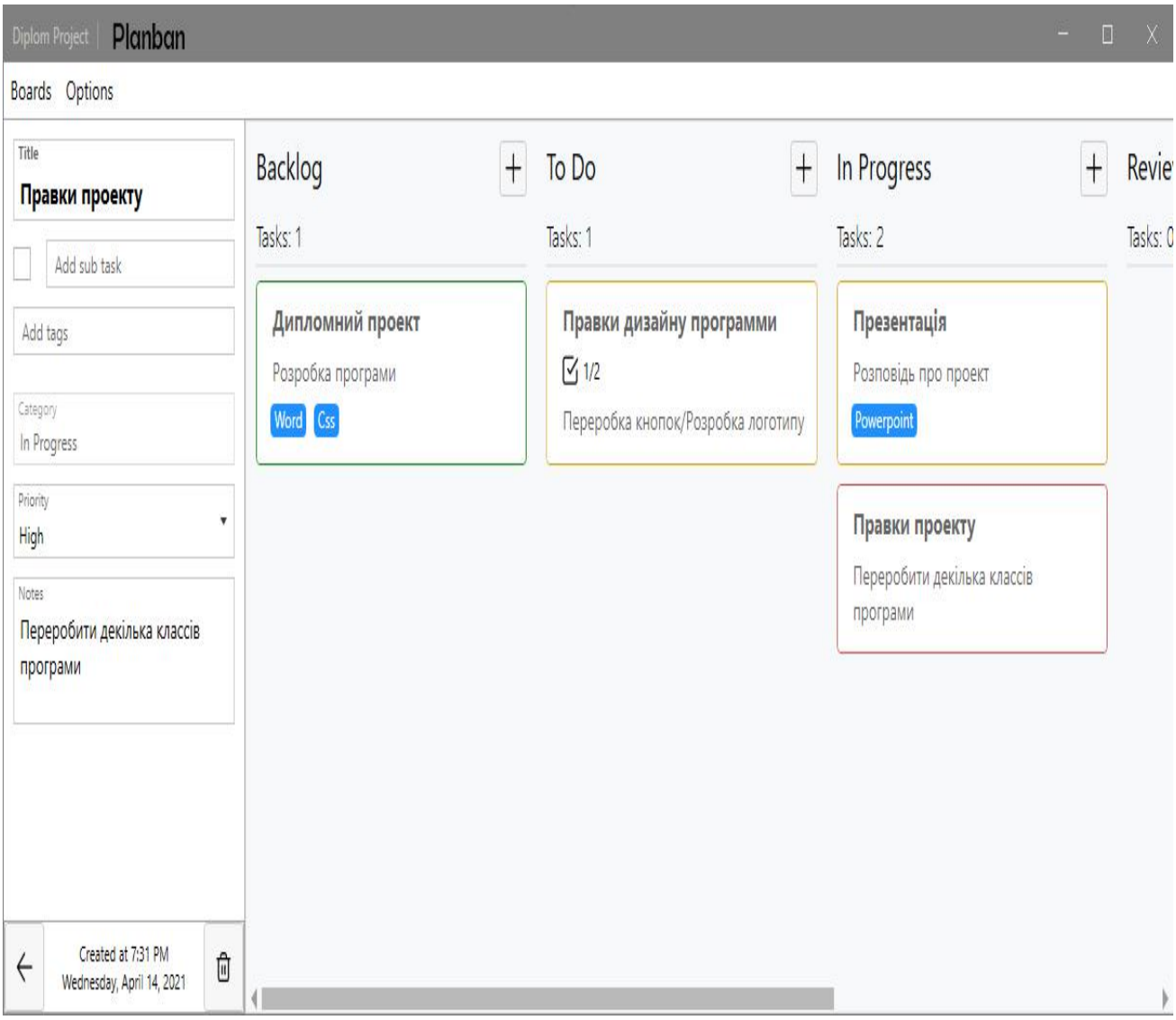

Рисунок 14 – Головне меню

Кожну задачу можна перетягувати у наступну колонку, для того щоб вона перейшла від початкової стадії проекту до його реалізації (рисунок 15). Всі її дані залишаються без змін. Колонка у якій була зачада зникає і одразу ж з'являється нове місце у наступній (рисунок 16).

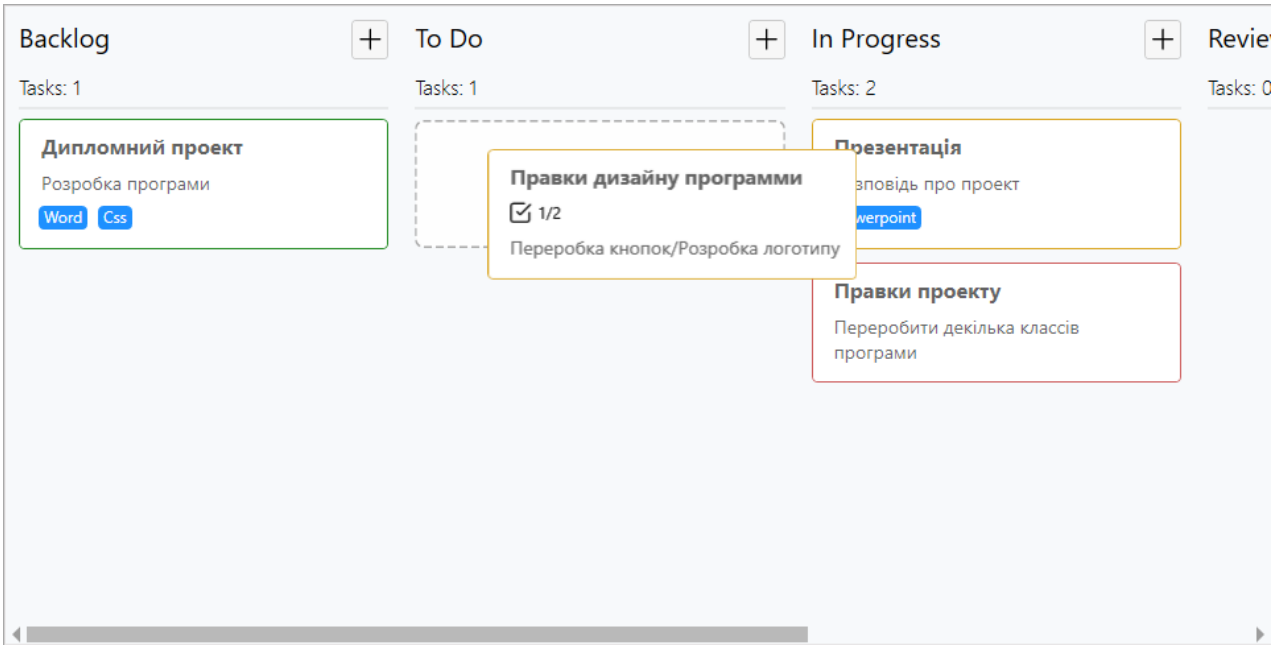

Рисунок 15 – Система управління дошкою

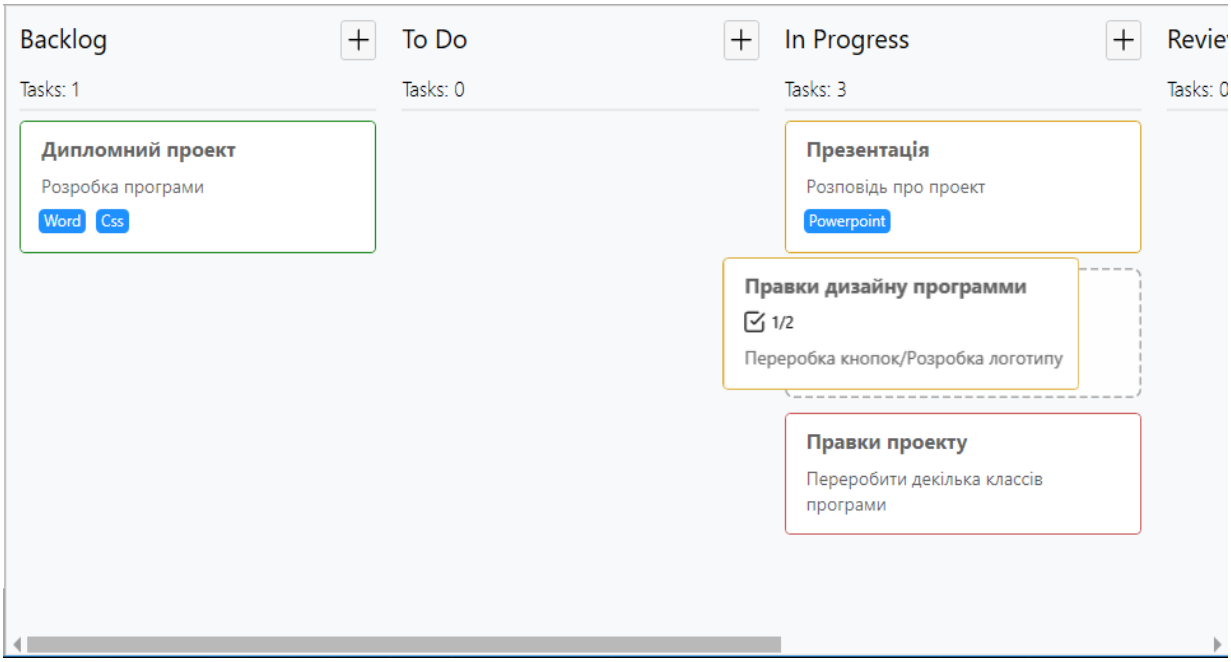

Рисунок 16 – Система зміни задачі

2.4.1 Технологія WPF

Windows Presentation Foundation — це презентаційний функціональний шар для системи Windows. Що робить його унікальним, і тому його слід розглядати як гідну альтернативу підсистемі Windows Forms. У той час як механізм рендеринга Windows Forms використовує двигуни GDI / GDI + bitmap, механізм WPF має свій

власний двигун векторного рендеринга. З цієї причини, він не створює вікна і елементи управління стандартним способом і стандартним чином, як це робить Windows. Технологія WPF радикально відрізняється від технології Windows.

У графічної підсистеми Windows Forms розробник, зазвичай, визначає користувальницький інтерфейс, за допомогою візуального дизайнера, автоматично генеруючи код (на мові проекту) в файлі .designer. Таким чином, по суті, призначений для користувача інтерфейс визначався і управлявся кодом на C# або VisualBasic.

У той же час призначений для користувача інтерфейс, створений за допомогою технології WPF, фактично визначається розширюваним мовою розмітки додатків, який зазвичай називають XAML (вимовляється як "заммель"). Ця мова, заснована на мові XML і спеціально розроблена компанією Microsoft для використання в системі WPF. Саме XAML, що лежить в основі WPF, надає їй міць і гнучкість, дозволяючи розробляти набагато багатші і складні інтерфейси, ніж Windows Forms.

Визначення, призначеного для користувача інтерфейсу за допомогою XAML, однаково для будь-якої мови проекту. З цієї причини, поряд з новими можливостями управління, яке є призначеним для користувача, з'явилася велика кількість нових допоміжних концепцій, які впливають на код. Наприклад, з'явилися властивості залежностей, значеннями яких є вираження, що часто потрібно в багатьох сценаріях зв'язування, для підтримки складних прив'язок, що надаються XAML. Однак код в додатку WPFK практично ідентичний коду в стандартному додатку Windows Forms, і розробнику слід сконцентруватися на присвятити мови XAML. При розробці WPFK-додатків необхідно мислити зовсім інакше, ніж при розробки додатків з використанням технології Windows Forms.

Розробник повинен зосередитися на перевагах нових можливостей зв'язування, XAML, і ставитися до свого коду не як до контролера призначеного для користувача інтерфейсу, а як до механізму обслуговування. Тепер код не повинен "проштовхувати" дані в призначений для користувача інтерфейс і

говорити йому, що робити. Замість цього для користувача інтерфейс повинен питати код, що робити, і посилати йому запити на дані ("виштовхувати" їх з нього). Різниця майже непомітна, але те, як ви визначаєте презентаційний шар вашого застосування, змінюється радикально. Призначений для користувача інтерфейс тепер є начальником. Код може (і повинен) приймати рішення, але він більше не повинен ініціювати дії.

WPF (Windows Presentation Foundation) WPF — це нова технологія .NET Framework3 для створення користувацьких інтерфейсів в клієнтських додатках. Одна з проблем, яка була вирішена при розробці та впровадженні WPF, — це поділ роботи між дизайнерами і програмістами. Рішення полягає в поділі вихідного коду програми WPF на дві частини: декларативне опис користувальницького інтерфейсу за допомогою XAML (Extensible Application Markup Language), код на мові програмування (такому як C#) що містить обробку подій. Для компіляції WPF-додатків зазвичай використовується Microsoft Build Engine (MSBuild) — технологія, що входить до складу .NET Framework у вигляді набору збірок.

Компіляція WPF-додатків при компіляції XAML файли аналізуються і перетворюються в коди Binary Application Markup Language (BAML), які вбудовуються в виконуваний файл в якості ресурсу. Код BAML більш компактний, ніж вихідний код XAML, і при виконанні він завантажується швидше, ніж файл XAML. При компіляції кожного файлу XAML генерується файл мови програмування, який містить часткове оголошення класу для елемента верхнього рівня в файлі розмітки. Як правило, файли XAML компілюються за два проходи. Спочатку компілюються тільки ті файли XAML, які не містять посилань на локально певні типи (тобто типи, певні в інших місцях проекту), оскільки вони існують тільки у вигляді вихідного коду і ще не були скомпільовані.

XAML — це файли з посиланнями на локальні змінні, певні типи цих файлів компілюються під час другого проходу компілятора. Вся необхідна для роботи MSBuild інформація про вихідні файлах, посиланнях на залежні збірки і

конфігурації програми знаходиться в файлах проекту MSBuild — XML файлах, що підкоряються схемою MSBuild.

XML (EXtensible Markup Language) — це простий гнучкий текстовий формат, який використовується як основа для створення мов розмітки, для публікації документів і обміну даними. Елементи документа можуть бути вкладеними, але вони не можуть перекриватися. Таким чином, елементи документа утворюють дерево. Кожен XML-документ повинен мати один і тільки один кореневий елемент. Вони можуть містити атрибути, які представляють собою пари ім'я-значення.

Кожен елемент документа повинен мати ім'я. Ім'я елемента — це рядок символів, яка починається з символу підкреслення або букви і складається тільки з букв, цифр, підкреслення, дефіса та точки. Імена елементів чутливі до регістру. Кожен з них починається з початкового тега і закінчується кінцевим тегом. Все, що знаходиться між відкриваючим і закриваючим тегами, називається вмістом елемента.

Текстові дані елемента XML — це або символьні дані (CDATA), або розібрані символьні дані, або їх комбінація. Визначено п'ять посилань на сутності, які можна використовувати в самому документі замість кутових дужок і символу слеш, щоб парсер XML не розглядав їх як керуючі символи розмітки.

Всі символи між комбінацією символів розглядаються парсером XML як вміст, що не включає символи розмітки. Всі інші змінні, є розібраними символьними даними, які парсер XML інтерпретує як символи розмітки. Посилання на символи не працюють всередині розділів CDATA. Розділи CDATA не можуть бути вкладеними. Елемент може мати атрибути, вони складаються з пар ім'я-значення. Імена атрибутів підкоряються тим же вимогам, що і імена елементів, а вони в свою чергу можуть міститися в відкриваючих тегах елемента. Плюси заключаються в тому, що кількість атрибутів елемента не обмежена, а порожній елемент може містити атрибути. Значення атрибута укладено в однакові лапки — одинарні або подвійні. Для різних атрибутів можуть використовуватися різні лапки.

ВИСНОВОК

На даний момент є багато різних методологій управління проектами, які використовуються у всьому світі. Існує безліч версій, яка методологія краща для розробки програмного забезпечення, але є такі, які пройшли випробування часом. Однією з таких є канбан, та його система управління проектами, за допомогою дошок зі стікерами.

Основною ідеєю було постійне відстежування проектів на кожному його етапі. При цьому підході процес від опису задачі до доставки результатів її виконання користувачу, наочно показується учасникам процесу, і члени команди можуть витягувати роботу з черги. Канбан в контексті розробки програмного забезпечення означає систему візуального управління процесом, яка вказує, що виробляти, коли виробляти і як багато виробляти і бере за основу ощадливе виробництво.

Отже в ході написання випускної кваліфікаційної дипломної роботи був розробленно готовий продукт, який можна використовувати за для роботи над проектоми в IT компаніях. В рамках роботи були виконані всі поставлені завдання. Було вивчено різні методології управління проектами, також проведено порівняння і аналіз переваг і недоліків, існуючих систем. Внаслідок чого був вибраний метод використання канбан-дошки.

Таким чином, результатом випускної кваліфікаційної роботи, є повністю реалізована програма для управління проектами та процесами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Scrum.org The home of scrum [Електронний ресурс] – Режим доступу: https://www.scrum.org/resources/what-is-scrum
2. Scrum: The Art of Doing Twice the Work in Half the Time J. Sutherland [Електронний ресурс] – Режим доступу: https://www.scruminc.com/new-scrum-the-book/
3. Kanban Explained for Beginners [Електронний ресурс] – 2019. – Режим доступу: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban
4. Kanban Boards: The Ultimate Guide [Електронний ресурс] – Режим доступу: https://www.projectmanager.com/kanban
5. How to Plan an Event: Event Plan Steps, Tips and Checklist [Електронний ресурс] – Режим доступу: https://www.projectmanager.com/training/how-to-plan-an-event
6. PMBOK® Guide – Sixth Edition [Електронний ресурс] – 2019. – Режим доступу: https://www.pmi.org/pmbok-guide-standards
7. Офіційний сайт Visual Studio [Електронний ресурс] – Режим доступу: https://visualstudio.microsoft.com
8. Офіційний сайт litedb [Електронний ресурс] – Режим доступу: https://www.litedb.org
9. What is a good choice of database for a small .NET [Електронний ресурс] – Режим доступу: https://stackoverflow.com/questions/6749556/what-is-a-good-choice-of-database-for-a-small-net-application
10. Офіційний сайт Microsoft [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/
11. Implementing Agile: Creating an Agile Environment [Електронний ресурс] – Режим доступу: https://www.agilealliance.org/agile-practice-guide/
12. Pmclub project manager assistant [Електронний ресурс] – Режим доступу: https://pmclub.pro/courses/ocenka-proektov-i-zadach

13. Security Analysis of End-to-End Encryption in Telegram [Електронний ресурс] / Jeeun Lee, Rakyong Choi, Sungsook Kim, Kwangjo Kim – Режим доступу: https://caislab.kaist.ac.kr/publication/paper_files/2017/SCIS17_JU.pdf
14. Messina, C. 2016 will be the year of conversational commerce [Електронний ресурс] / C. Messina // Medium. - 2016. - Режим доступу: https://medium.com/chris-messina/2016-will-be-the-year-of-conversational-commerce-1586e85e3991
15. Jira Software project manager assistant [Електронний ресурс] – Режим доступу: https://www.atlassian.com/ru/software/jira
16. How to design a logo: the ultimate guide Antonia Gesch [Електронний ресурс] – Режим доступу: https://99designs.com/blog/logo-branding/how-to-design-logo/
17. .NET Foundation "State of .NET" [Електронний ресурс] – Режим доступу: https://dotnetfoundation.org
18. Windows Presentation Foundation documentation [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-5.0