# Програмне та ТЕХНічне забезпечення

## **Засоби розробки**

При створенні програмного продукту середовищем розробки було обрано Visual Studio 2010 для програмування мовою C#, використовуючи плагін ReSharper.

Дане середовище дозволяє розробляти як консольні застосунки, так і застосунки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторингу коду. Вбудовані інструменти включають в себе редактор форм для спрощення створення графічного інтерфейсу програми, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні доповнення (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду, додавання нових наборів інструментів.

Плагін ReSharper дуже зручно використовувати при розробці мовою C#. Він проводить статичний аналіз коду (пошук помилок в коді до компіляції) в масштабі всього рішення, передбачає додаткові засоби автозаповнення, навігації, пошуку, підсвічування синтаксису, форматування, оптимізації та генерації коду, надає 40 автоматизованих рефакторингів.

C# – сучасна мова програмування, яка має дуже багато переваг перед такими мовами, як C/C++ або Java. Ось неповний їх список:

* C# створювався паралельно з платформою .NET Framework і повною мірою враховує всі його можливості;
* C# є повністю об'єктно-орієнтованою мовою, де навіть типи, вбудовані в мову, представлені класами;
* C# є потужною об'єктною мовою з можливостями наслідування і універсалізації;
* C# є спадкоємцем мов C/C++, зберігаючи кращі риси цих популярних мов програмування. Спільний з цими мовами синтаксис, знайомі оператори мови полегшують перехід програмістів від С++ до C#;
* зберігши основні риси C++, мова стала простіше і надійніше. Простота і надійність, головним чином, пов'язані з тим, що на C# хоча і допускаються, але не заохочуються такі небезпечні властивості С++ як покажчики, адресація, розіменування, адресна арифметика;
* завдяки платформі .NET Framework, що стала надбудовою над операційною системою, програмісти C # отримують ті ж переваги роботи з віртуальною машиною, що і програмісти Java. Ефективність коду навіть підвищується, оскільки виконавча середа CLR є компілятором проміжної мови, в той час як віртуальна Java-машина є інтерпретатором байт-коду;
* потужна бібліотека платформи підтримує зручність побудови різних типів застосунків на C#, дозволяючи легко будувати Web-служби, інші види компонентів, достатньо просто зберігати і отримувати інформацію з бази даних.

Для збереження інформації було обрано систему управління реляційними базами даних Microsoft SQL Server 2008 R2.

Це сучасна відмовостійка СУБД, яку дуже зручно використовувати при розробці застосунків на основі платформи .NET Framework. Платформа надає спеціальні засоби для роботи з Microsoft SQL Server.

Ця СУБД використовується для роботи з базами даних розміром від персональних до великих баз даних масштабу підприємства.

В якості системи контролю версій було використано розподілену систему керування версіями файлів Git.

Використання розподіленої системи контролю версій дає наступні переваги (над централізованими):

* періодична синхронізація декількох комп'ютерів під управлінням одного розробника (робочого комп'ютера, домашнього комп'ютера, ноутбука і так далі). Використання розподіленої системи позбавляє від необхідності виділяти один з комп'ютерів в якості сервера, а синхронізація виконується за потребою;
* спільна робота над проектом невеликою територіально розподіленою групи розробників без виділення загальних ресурсів. Як і в попередньому випадку, реалізується схема роботи без головного сервера, а актуальність репозиторіїв підтримується періодичними синхронізації за схемою «кожен з кожним».
* великий розподілений проект, учасники якого можуть довгий час працювати кожен над своєю частиною, при цьому не мають постійного підключення до мережі. Такий проект може використовувати централізований сервер, з яким синхронізуються копії всіх його учасників. Можливі й більш складні варіанти - наприклад, зі створенням груп для роботи за окремими напрямками всередині більшого проекту. При цьому можуть бути виділені окремі «групові» сервери для синхронізації роботи груп, тоді процес остаточного злиття змін стає деревовидним: спочатку окремі розробники синхронізують зміни на групових серверах, потім оновлені репозиторії груп синхронізуються з головним сервером. Можлива робота і без «групових» серверів, тоді розробники однієї групи синхронізують зміни між собою, після чого будь-який з них (наприклад, керівник групи) передає зміни на центральний сервер.

Використання Git дає наступні переваги (над аналогічними розподіленими системами контролю версій):

* висока продуктивність;
* продумана система команд;
* репозиторії git можуть поширюватися і оновлюватися загальносистемними файловими утилітами резервного копіювання та оновлення, завдяки тому, що фіксації змін і синхронізації не змінюють існуючі файли з даними, а лише додають нові (за винятком деяких службових файлів, які можуть бути автоматично оновлені за допомогою наявних у складі системи утиліт);
* для поширення репозиторію по мережі досить будь-якого веб-сервера.

## **Вимоги до технічного забезпечення**

### **Загальні вимоги**

Головною метою розробки даного програмного продукту є обробка даних про складські приміщення, об’єми товарів і замовлень з метою отримання плану перевезень між складами і споживачами. Він призначений для використання співробітником підприємства, що працює у відповідному напрямі.

Для можливості роботи з даним програмним продуктом до складу технічних засобів повинні входити:

1. комп’ютер з наступною конфігурацією:

1) процесор з тактовою частотою не нижче 1.4 ГГц;

1. об’єм оперативної пам’яті не менше 128 МБ;
2. наявність 20 Мб вільного простору на жорсткому диску;
3. графічна карта з об’ємом вбудованої пам’яті не менше 32 Мб;
4. додатково має бути встановлене наступне програмне забезпечення:
5. операційна система Windows XP/Vista/7;
6. СУБД Microsoft SQL Server 2008 R2;
7. платформа .NET Framework 4.0;
8. комп’ютерна периферія, до складу якої входить:
9. монітор;
10. мишка;
11. клавіатура.

## **Архітектура програмного забезпечення**

### **Діаграма класів**

На рисунку 4.1 представлена структурна схема класів, які відповідають за виконання таких функцій програми, як встановлення з’єднання з СУБД Oracle, робота з даними, робота з транзакціями, складання та прогнозування плану випуску продукції та формування звітів «План виробництва».

Діаграма містить п’ять класів, а саме:

* «MainForm» - головний клас програми, відповідає за взаємодію програми з базою даних та містить основні функції роботи програми;
* «PasswordForm» - клас, що відповідає за коректне з’єднання з базою даних та запускає головний модуль програм;
* «ParametrPrograming» - клас у якому містяться функції, що виконують складання та прогнозування плану випуску продукції кожного виду;
* «ForecastPrice» - клас функції якого виконують прогнозування ринкових цін;
* «CreateReport» - клас призначенням якого є генерація звітів.

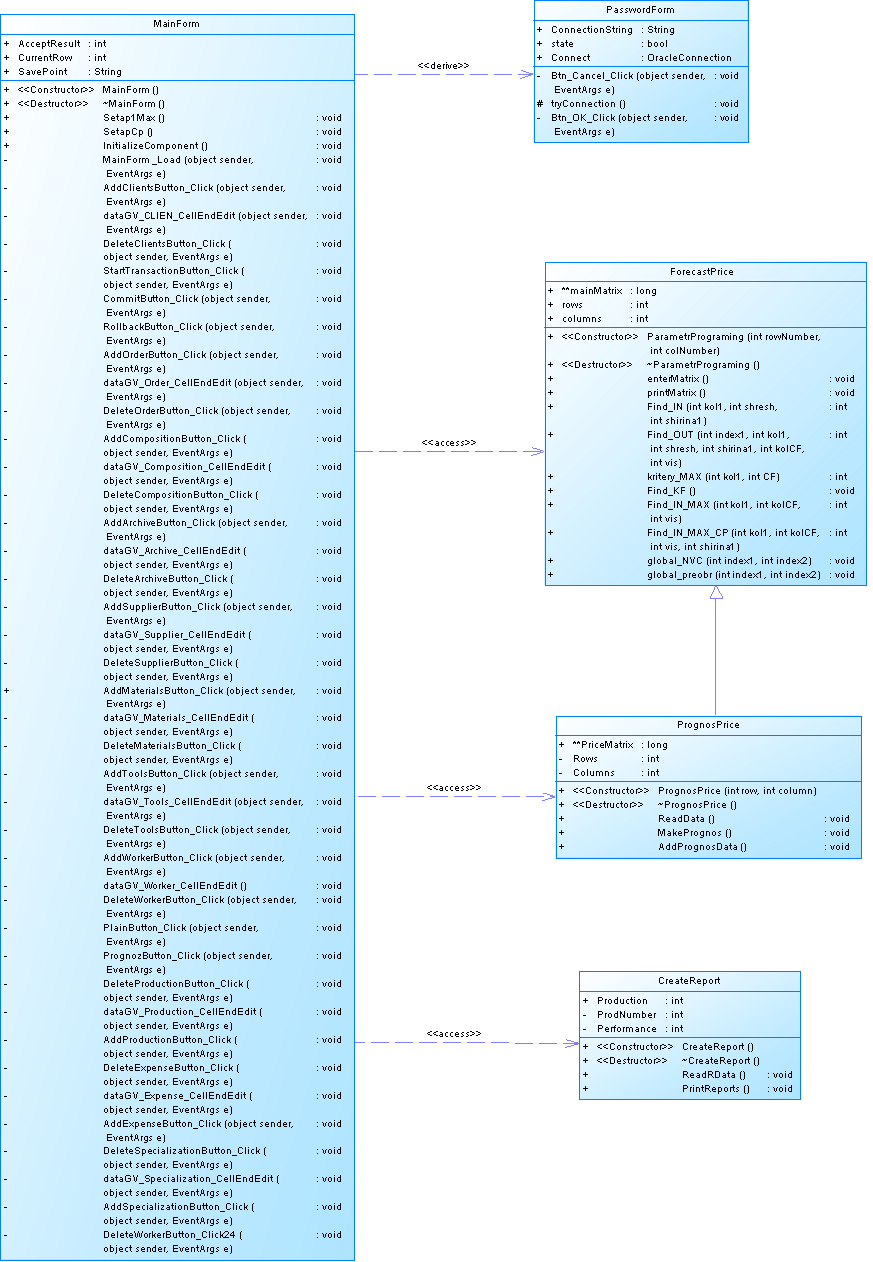


Рисунок ‎4.1 – Структурна схема класів

### **Діаграма послідовності**

На рисунку 4.2 представлена схема структурна послідовності. На даній діаграмі представлена типова послідовність дій, які виконуються при складенні оптимального плану виготовлення продукції, надання підприємству прогнозів плану виготовлення продукції на майбутні періоди та формуванні звіту «План виробництва». Також визначено акторів та приналежність їм відповідних дій, необхідних для виконання поставленої задачі.

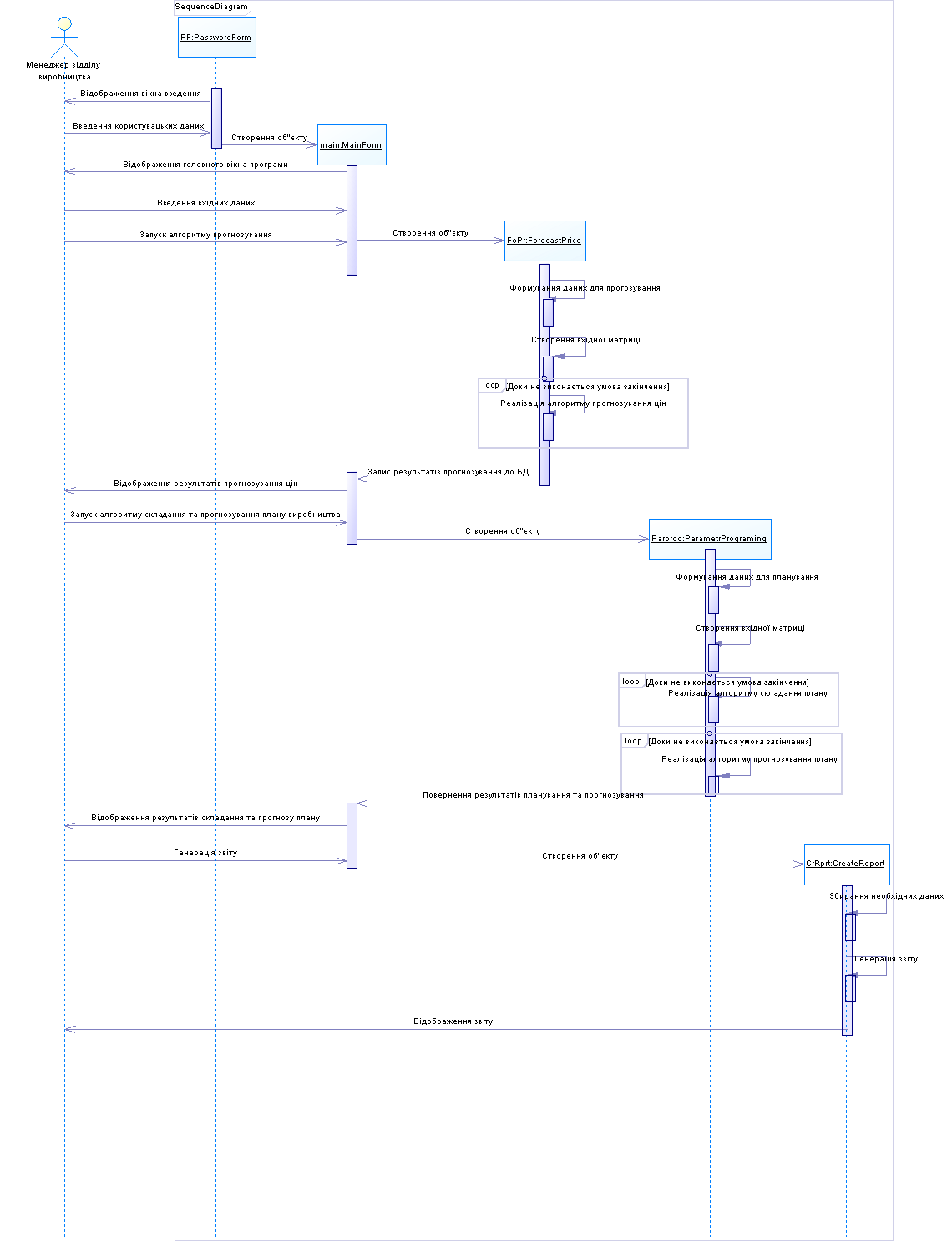


Рисунок ‎4.2 – Схема структурна послідовності

Далі, за допомогою діаграм послідовності опишемо дії, які виконуються при складанні плану випуску продукції та його прогнозуванні, а також при прогнозуванні ринкових цін.

На рисунку 4.3 представлена схема структурна послідовності дій роботи алгоритму. На даній діаграмі представлена типова послідовність дій, які виконуються при виконання алгоритму планування та прогнозування.



Рисунок ‎4.3 - Схема структурна послідовності дій роботи алгоритму планування та прогнозування

На рисунку 4.4 представлена схема структурна послідовності дій роботи алгоритму, який використовується при прогнозуванні ринкових цін за ретроспективними даними. На даній діаграмі представлена типова послідовність дій, які виконуються при виконання алгоритму прогнозування ринкових цін.

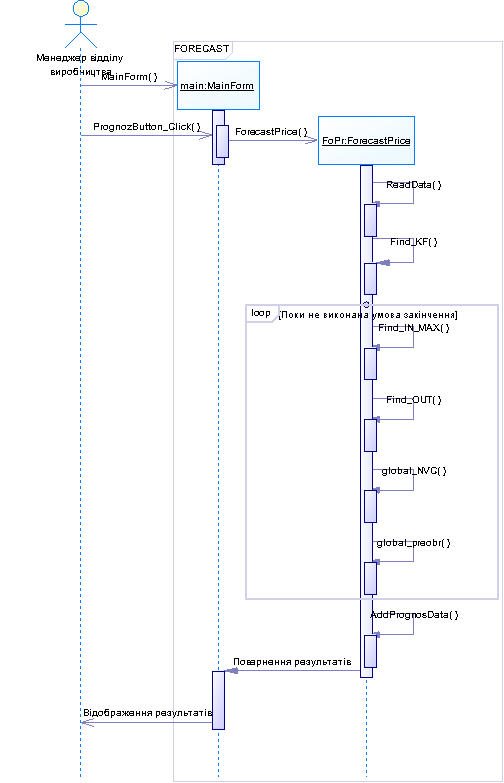


Рисунок ‎4.4 - Схема структурна послідовності дій роботи алгоритму прогнозування ринкових цін

### **Діаграма компонентів**

На рисунку 4.5 представлена схема структурна компонентів, на якій відображені компоненти, що використовуються в комплексі задач, та взаємозв’язки між ними. Основними компонентами в системі є співробітник підприємства, клієнтське застосування та СУБД Oracle.



Рисунок 4.5 – Схема структурна компонентів

### **Специфікація функцій**

Функції класів програмного забезпечення наведені в таблиці 4.9.

Таблиця 4.9 – Функції класів програмного забезпечення

| **Клас** | **Функція** | **Опис функції** |
| --- | --- | --- |
| Algorithm – абстрактний клас алгоритму, в якому реалізована спільна для ітеративних алгоритмів логіка | public abstract void DrawNodes() | Функція, яка реалізовує графічне відображення вузлів |
| public virtual string Info() | Функція повертає специфічну інформацію про поточний стан алгоритму |
| protected abstract void InnerIteration() | Специфічна ітерація для певного алгоритму |
| private void Iteration() | Проведення специфічної ітерації а також збір загальної статистики по алгоритму, ведення логу |
| public void Iterations (int count = 1) | Проведення певної кількості ітерацій |
| public void IterateToStop() | Виконання ітерації доки не буде досягнуто критерію зупину |
| private void OpenLogFile() | Відкриття лог-файлу для дозапису |
| public void SetNodes (List<Node> nodesForSet) | Задання вузлів для обробки у алгоритмі |
| public void Stop() | Зупинка алгоритму |
| BeesColony – клас-нащадок класу Algorithm. Клас, в якому реалізовано бджолиний алгоритм в загальному вигляді | private Site CreateNewSite() | Створює випадковий розв’язок задачі |
| public void CreateSites() | Генерація початкового набору розв’язків |
| public override void DrawNodes() | Функція, яка реалізовує графічне відображення вузлів |
| public override string Info() | Повертає інформацію про поточний стан алгоритму |
| protected override void InnerIteration() | Ітерація бджолиного алгоритму |
| Site – абстрактний клас, який представляє розв’язок бджолиного алгоритму а також реалізовує спільну логіку пошуку сусідів і переходу до кращого | public virtual int CompareTo(object obj) | Функція порівняння даного об’єкту з іншим |
| public void DrawNodes() | Графічне відображення |
| private List<Site> GenerateNeighbours (int count) | Генерація сусідів |
| protected abstract Site GetNeighbour() | Функція генерації сусіда |
| public bool GoToBestNeighbour (int countOfNeightbours) | Перехід до кращого сусіда, якщо такий є |
| protected abstract void GoToNeighbour (Site site) | Перехід до сусіда |
| public abstract List<Node> PrepareToDraw (Color connectionsColor) | Підготувати вузли до графічного відображення |
| SiteVrpTsp – клас-нащадок класу Site. Клас розв’язку задачі VRP, що зводиться до TSP. | private void ConnectConsumerToDepot (Node node1, Node node2, Color connectionColor) | Графічно під’єднати вузол-споживач до вузла-складу. |
| private static double[,] GeneratePricesByPositions (List<Node> nodes, int depotsCount, int consumersCount) | Згенерувати таблицю відстаней |
| private void GenerateSequence (int depotsCount, int clustersCount, int consumersCount) | Згенерувати послідовність обходження вузлів |
| private static double GetDistance (Node node1, Node node2) | Обчислення відстані від одного вузла до іншого |
| protected override Site GetNeighbour () | Згенерувати сусіда |
| protected override void GoToNeighbour (Site site) | Перейти до сусіда |
| private static void Interchange (int[] arr, int i1, int i2) | Поміняти місцями елементи масиву |
| private static void Invert (int[] arr, int i1, int i2) | Інвертувати частину масиву |
| public void InvertRandomPartOfNodesSequence () | Інвертувати випадкову частину масиву |
| public override List<Node> PrepareToDraw (Color connectionsColor) | Підготувати вузли до графічного відображення |
| public override string ToString() | Функція повертає відображення розв’язку у вигляді рядку |
|  |  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| private void AddClientsButton\_Click(object sender, EventArgs e) | Застосовується для додавання даних до таблиці клієнтів |
| private void dataGV\_CLIEN\_CellEndEdit(object sender, DataGridViewCellEventArgs e) | Застосовується для редагування даних в таблиці клієнтів |
| private void DeleteClientsButton\_Click(object sender, EventArgs e) | Застосовується для видалення даних з таблиці клієнтів |
| private void StartTransactionButton\_Click(object sender, EventArgs e) | Застосовується для початку транзакцій у базі даних |
| private void CommitButton\_Click(object sender, EventArgs e) | Застосовується для підтвердження здійснених змін у базі даних |
| private void RollbackButton\_Click(object sender, EventArgs e) | Застосовується для відміни останньої виконаної операції |
| private void AddOrderButton\_Click(object sender, EventArgs e) | Застосовується для додавання даних до таблиці договорів |

## **Опис звітів**

За допомогою програми можливий збір статистичних даних для побудови звітів «План виробництва».

Звіти створюються за допомогою відкритого COM-додатка, для роботи з сервером Microsoft Word. Даний додаток дозволяє посилати досить великий набір команд серверу Word і створювати звіти будь-якої складності та деталізації. При написанні коду будемо використовувати .Net збірку взаємодії з програмами Microsoft Office – Microsoft.Office.Interop.Word [12], для роботи нам потрібно лише одна частина даної збірки, а саме Word.Application. Головним в ієрархії об’єктів Word.Application є об’єкт Document. Інформація про об’єкти Document зберігається у вигляді посилань на відкриті документи у властивості Document. Document – це пустий лист, на якому генерується звіт «План виробництва».

Звіт «План виробництва» генерується автоматично при виборі в програмі пункту меню «Згенерувати звіт» і зберігається під назвою «План.doc» в папці Reports, що знаходиться в кореневій папці. Згенерований звіт має вигляд зображений на рисунку 4.6, де показано …

<Тут має бути зразок згенерованого звіту>

Рисунок 4.6 – Вигляд звіту «План виробництва»

## **Висновок до розділу**

*Дати висновок до цього розділу.*