Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики і обчислювальної техніки

Кафедра технічної кібернетики

**Звіт**

З переддипломної практики на тему:

«Автоматизована підсистема обрахунку стажів працівників КПІ ім. Ігоря Сікорського»

Місце проходження практики:

КБІС КПІ ім. Ігоря Сікорського

Виконав

студент 4 курсу ФІОТ,

групи ІК-41

Літвін І.Д.

Керівник дипломного проекту:

Максимюк А.В.

(підпис)

Дата захисту Оцінка

Київ – 2018

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

Інститут (факультет): інформатики та обчислювальної техніки

Кафедра: технічної кібернетики

Напрям підготовки: 6.050201 **Системна інженерія**

**ЗАВДАННЯ**

**на практику студенту**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Літвіну Іллі Денисовичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема проекту: Автоматизована підсистема обрахунку стажів працівників КПІ ім. Ігоря Сікорського

Керівник проекту: Максимюк А.В.

2. Завдання на практику:

2.1 Розробка моделі бізнес-процесів на основі вже існуючих рішень.

2.2 Розробка алгоритмічної структури програми на основі раніше створеної моделі бізнес-процесів.

2.3 Оптимізація алгоритмічної структури на основі моделі бізнес-процесів.

2.4 Розробка графічних інтерфейсів програми. Написання коду програми та тестування.

2.5 З’єднання всіх модулів та компонентів для отримання цілісної програми.

3. Дата видачі завдання: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Літвін І.Д.\_\_\_\_

(підпис) (ініціали, прізвище)

Керівник проекту \_\_\_\_\_\_\_\_\_\_\_\_ Максимюк А.В.

(підпис) (ініціали, прізвище)

Зміст

1. Розробка моделі бізнес-процесів на основі вже існуючих рішень

2. Розробка алгоритмічної структури програми на основі раніше створеної моделі бізнес-процесів

3. Оптимізація алгоритмічної структури на основі моделі бізнес процесів

4. Розробка графічних інтерфейсів програми. Написання коду програми та тестування

5. З’єднання всіх модулів та компонентів для отримання цілісної програми

**1. Розробка моделі бізнес-процесів на основі існуючих рішень**

Завданням на переддипломну практику являється розробка автоматизованої підсистеми обрахунку стажів працівників КПІ ім. Ігоря Сікорського.

Для моделювання бізнес-процесів використовується декілька різних

методів, в основі яких лежить як структурний, так і об'єктно-орієнтований підходи до моделювання. Проте, класифікація самих методів на структурні та об'єктні є доволі умовною, оскільки найбільш розвинуті методи використовують елементи обох підходів. Стисло розглянемо характеристики найбільш поширених методів:

* метод функціонального моделювання SADT (IDEF0);
* метод моделювання процесів IDEF3;
* моделювання потоків даних DFD;
* метод ARIS;
* метод Ericsson-Penker;
* метод технології Rational Unified Process.

Метод моделювання IDEF3 - частина сімейства стандартів IDEF; використовується для моделювання послідовності виконання дій і їх взаємозалежностей в рамках процесу. Метод отримав визнання серед системних аналітиків як доповнення до методу функціонального моделювання IDEF0.

Основою моделі IDEF3 служить сценарій процесу, який відокремлює послідовність дій і підпроцесів системи. Як і в методі IDEF0, основною одиницею моделі є діаграма. Іншим важливим компонентом є дія або "одиниця роботи" (Unit of Work), взаємодія яких зображається за допомогою зв'язків.

Діаграми потоків даних (Data Flow Diagrams - DFD) представляють собою ієрархію функціональних процесів, що пов'язані потоками даних. Мета такого представлення полягає у демонстрації того, як кожен процес перетворює свої вхідні дані у вихідні і виявлення зв'язків між цими процесами.

Відповідно до методу, модель системи визначається як ієрархія діаграм потоків даних, основними компонентами яких є:

* зовнішні об'єкти;
* системи та підсистеми;
* процеси;
* накопичувачі даних;
* потоки даних.

Перший компонент представляє собою матеріальний об'єкт або фізичну особу, яка є джерелом або приймачем інформації; наприклад: замовники, персонал, постачальники, склад.

**2. Розробка алгоритмічної структури програми на основі раніше створеної моделі бізнес процесів**

Основні властивості алгоритмів:

Скінченність. Алгоритм має завершуватися за скінченну кількість кроків (скінченність процесу перетворення інформації).

Результативність (завершеність). При коректно заданих вихідних даних виконання алгоритму завжди повинно приводити до певного результату.

Дискретність. Алгоритм повинен представляти процес вирішення завдання як послідовне виконання деяких простих кроків, які виконуються послідовно і за скінчений час.

Визначеність (детермінованість або однозначність). Дії, які необхідно виконати на кожному кроці алгоритму, повинні бути чітко та недвозначно визначені для кожного можливого випадку. Порядок виконання операцій має бути строго визначеним. Це означає, що виконання алгоритму відбувається у єдиний спосіб та призводить до однакового результату для однакових вхідних даних.

Формальність. Будь-які виконавці, здатні сприймати і виконувати вказівки алгоритму (навіть не розуміючи їх змісту), діючи за алгоритмом, можуть виконати поставлене завдання і отримати однакові результати.

Масовість (універсальність). Алгоритм може бути використаний для розв’язання цілого класу однотипних задач при різних наборах вихідних даних (наприклад, квадратного рівняння з різними коефіцієнтами).

Зрозумілість. Алгоритм повинен складатися з команд, які входять до системи команд його виконавця. При цьому виконавцем алгоритму може бути людина, комп’ютер, робот тощо.

З урахуванням даних властивостей поняття алгоритму часто визначається як скінченна однозначно визначена послідовність операцій, формальне виконання яких приводить до розв’язання певної задачі за кінцеве число кроків.

З поняттям алгоритму пов’язані такі поняття, як область його задання, складність, еквівалентність, алгоритмічна розв’язність та ін.

Область задання алгоритму — це множина даних, до яких алгоритм застосовний. Якщо алгоритм завершується без отримання результату або продовжується необмежено довго, то говорять про незастосовність алгоритму до цих вхідних даних.

Під алгоритмічною розв’язністю розуміють можливість побудови алгоритму розв’язку всіх задач даного класу.

Існують класи задач, для розв’язання яких не існує єдиного універсального способу. Це алгоритмічно нерозв’язувані проблеми. Для визначення алгоритмічної розв’язності якогось класу задач необхідно або побудувати алгоритм розв’язку, або довести неможливість побудови такого алгоритму (довести, що проблема є алгоритмічно нерозв’язною). Наприклад, алгоритмічно розв’язна проблема — доведення тотожностей в алгебрі (відомі правила перетворення алгебраїчних виразів). У той же самий час розв’язання диференційних рівнянь — проблема алгоритмічно нерозв’язна. Є проблеми, про які невідомо, чи є вони алгоритмічно розв’язні, чи є алгоритмічно нерозв’язні.

**3. Оптимізація алгоритмічної структури на основі моделі бізнес процесів**

Очевидно, найбільш природнім способом звуження інтервалу невизначеності для одновимірної унімодальної функції є ділення його на декілька рівних частин з наступним обчисленням значень цільової функції в вузлах отриманої сітки (рис. 13.6).

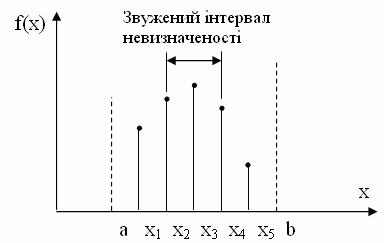


Рисунок 13.6 – Метод загального пошуку

В результаті інтервал невизначеності звужується до двох кроків сітки. Звичайно говорять про дроблення інтервалу невизначеності, яке характеризується коефіцієнтом f. Розділивши інтервал невизначеності на N частин, отримаємо N+1 вузол.

Щоб отримати значення f = 0,01, необхідно обчислити цільову функцію в 199 точках, а при f = 0,001 N=1999. Звідси видно, що ефективність цього методу при зменшенні інтервалу невизначеності швидко падає. Напрошується інший варіант: щоб отримати f=0,01, необхідно обчислити спочатку функцію в 19 точках і отримати f = 0,1, а потім обчислити ще 19 значень функції на скороченому інтервалі невизначеності, отримати f = 0,01, зробивши при цьому всього 38, а не 199 обчислень. Таким чином, при деякій винахідливості ефективність пошуку можна різко збільшити.

**4. Розробка графічних інтерфейсів програми. Написання коду програми та тестування**

Приймальне тестування (Acceptance testing)

Це тестування, спрямоване на те, щоб зробити висновок придатний наш додаток до використання або варто щось допрацювати або виправити. Таке тестування проводиться зазвичай на стороні замовника після закінчення розробки і тестування функціоналу. Тести пишуться або придумуються і виконуються зазвичай самим замовником.

Функціональне тестування (Functional testing)

Функціональне тестування - це тестування функціональності і поведінки нашої програми, для того щоб переконатися що поведінка програми та її функціонал відповідає вимогам функціональної специфікації. Зазвичай виконується як тестування чорного ящика, подаючи на вхід якийсь набір даних і чекаючи чогось на виході.

Смоук тестування (Smoke testing)

Тестування основних модулів додатка з метою визначення чи придатна програма для релізу нової версії. Звичайно проводиться перед великим релізом нової версії додатка, який включає в себе нові модулі. Проводиться для переконання що нові модулі справно працюють і що старі модулі і функціональність теж не зламалася і працює вірно.

Тестування Безпеки (Security testing)

Перевірка прав доступу, неможливість заходу в програму стороннім особам, SQL ін'єкції, XSS атаки. Перевірка що додаток відповідає необхідним вимогам до безпеки.

Стрес Тестування (Stress testing)

Це тестування програми або окремого модуля на межі або вище його можливостей. Проводиться для визначення цієї межі і поведінку при переході цієї межі. Отримати збій і проаналізувати чи достатньо цього кордону на поточний момент розвитку проекту додатка.

Юзабіліті Тестування (Usability testing)

Перевірка з якою легкістю користувач зможе вивчити наш додаток і користуватися ним. Проводиться для оптимізація, поліпшення і спрощення інтерфейсу до інтуїтивно зрозумілого. Сюди можуть входити всілякі підказки, хелпери, зручний дизайн елементів, мінімізація кроків проходження для доступу до функції і т. д.

**5. З’єднання всіх модулів та компонентів модуля для отримання цілісної програми**

Збірка в термінології .NET- двійковий файл (exe або dll), що містить номер версії інформаційного продукту, метадані та інструкції для віртуальної машини .NET. Файл з вихідним кодом, написаному на будь-якій мові підтримуваної .NET платформою (C #, VB.NET ...), за допомогою відповідного компілятора (C # компілятор, VD.NET компілятора ...) компілюється в збірку. Номер версії ІП складається з довільної рядки і чотирьох цілих чисел, основний і додатковий номера версії, номер збірки і номер редакції.