South China University of Technology

《Software Engineering》Experiment Report

Experiment Title： Experiment 2: Prototype and software testing

Name： 谭演锋 杨竣尧 Student ID：202130100456 202130442068

Class： 计算机全英联合班 Group：

Collaborator：

Teacher： 李剑

|  |
| --- |
| **Description** |
| 【Objective and Requirement】  Objective：  The public works department of a large city is embarking on the development of an online Pothole Tracking and Repair System (PHTRS). The system allows citizens to log in and report potholes, specifying the location and severity. As potholes are reported, they are recorded in a "Public Works Department Repair System" and assigned a unique identifier. The system stores information such as street address, size (rated on a scale of 1 to 10), location (middle, curb, etc.), district (derived from the street address), and repair priority (determined by the size of the pothole).  Each reported pothole is associated with work order data, including details such as pothole location and size, repair crew identification, crew size, assigned equipment, repair hours, hole status (work in progress, repaired, temporarily repaired, not repaired), amount of filler material used, and repair cost (calculated from hours applied, crew size, materials, and equipment used). Additionally, a damage file is created to store information about reported damages caused by the pothole. This file includes the citizen's name, address, phone number, type of damage, and the dollar amount of damage. PHTRS operates as an online system, and all queries are conducted interactively.  Requirement：   1. Development: Implement a prototype of PHTRS system  * Include user interfaces and one function * Implemented in Ruby (Rails) or other object-oriented programming languages (or Java, C++) * Object-oriented * MVC(Model-View-Controller) architecture pattern  1. Test:  * Draw a graph to show the structure (execution) of a selected procedure in your program (or your Ruby program). * Compute the complexity of this procedure according to the structure graph  1. Test case design: Design test case for the selected procedure  * Should cover all statements (or methods) of the procedure * Every branch should be exercised for true and false conditions * List inputs and corresponding execution paths   【Environment】  Operating System：Windows 10 & MacOS |
| **Content** |
| 【Procedure】  **Experiment 2.1**  Our PHTRS is basing on java, Spring boot(Backend) and Vue(Frontend) for implementation.  The implementation of MVC:    Fig. The MVC   1. **Model:**    * Manages pothole data, including locations, statuses, and repair priorities.    * Handles business logic and interacts with the database for data consistency.   We have 2 models User and Hole    Fig. Model files  **User**: id, userName, password, userType, name, mobile, address. It is bonding to the table user in database phtrs by using the annotation.    Fig. User Model  **Hole**: id, wokers\_id, cost, username, priority, address, size, material, district, position, time, device, status. They are stored in the table Hole.    Fig. Hole Model  We use the Mysql for database and connecting them by jdbc.    Fig. Command for Connecting DB   1. **View:**    * Presents the user interface, displaying pothole information and a map of pothole locations.    * Allows users to report new potholes, update information, and view repair progress.   We create view by Vue framework, including the page of login, register, page for citizens, page for workers and page for admins. We also create the page for them to modify their self information.  Following is the file structure of views. Component are the views show to the users. The index.js in router is used to mapping the relationship between Controller and the Views.    Fig. Views Files  **Login View and Register View:**  User can login using their suing name and password. Login send the request, then controller (Back end) will return different page according to the user type in the data base.    Fig. Login views  User can register their account. However then cannot register into the admin. We also giving a judgement of the strength of password. The page can give a warn on some errors such as need username. the compare between two password.    Fig. Register View  **Citizen View:**  Citizens can see where the holes are in the page and its status whether they are fixing or unfixed in order to monitor the government. Citizens can report the situation of Hole by pressing the button New Record.  **IMG_0877**  Fig. Citizen View for hole.  Following is the form for citizens add new hole records . Citizens should provide the hole address, and select the size of hole (1-10), where is the hole and the Priority corresponding the size(1-3 low, 4-6 medium, 7-10 high). The other columns in hole table will be added autonomous by systems.  **IMG_0876**  Fig. View for adding hole.  **Worker View:**  Worker can see the details of holes including the worker who takes responsibilities for them and the number of workers fixing the holes. They can set many details of the hole by pressing the “Modify” button. If they finishing the the worker can press the button “Finished” to ask for admin to verify the hole has been fixed. Then the worker can not modify the hole.  **/Users/ha/Library/Containers/com.kingsoft.wpsoffice.mac/Data/tmp/photoeditapp/20231221175418/temp.pngtemp**  Fig. Part of Worker View for hole.  Workers can add the worker ID , the equipment, material material for the holes. They can also change the status of hole to inform others. They can set the cost and hour of hole.  **IMG_0875**  Fig. View for modifying the details of hole.  **Admin View:**  The Admin can see who report the holes and the report person information. Then can confirm whether the holes is fixed. Once the hole is fixed, the record will be deleted from the database  /Users/ha/Library/Containers/com.kingsoft.wpsoffice.mac/Data/tmp/photoeditapp/20231221175445/temp.pngtemp  Fig. Part of Admins View for hole.  After pressing the “Confirm” button, we can see the following view. The admin can select whether the fixing is pass or not. If it is not pass, the worker can modify the hole again.  IMG_0882  Fig. Confirm view   1. **Controller:**    * Acts as an intermediary, processing user inputs from the View and updating the Model accordingly.    * Enforces business logic and ensures consistency between the Model and View.   We have two main controllers for Login and Hole.    **loginContraoller:**  Following figure is an example of code in login controller. We use annotation RequstMapping to router the request of “Login” to the controller. Then the controller use the form data in the Views and identify the correctness of login by calling the “LoginDAO” which is the file with codes about sql queries that request the database. If the database has the corresponding record, the user will be got. Then the controller return true and the UserType (Model function) for showing the right View for different type of User.    **holeContraoller:**  Following figure is the figure about CRUD of the Hole. They all call the function of holeDAO to implement the SQL query and return OK to the View for the view updating.    **Experiment 2.2**  We will do the test on the following function:   |  | | --- | | #include <iostream>  int main() {  char continueFlag;  do {  int number;  // Step 1: 输入一个数字  std::cout << "请输入一个整数：";  std::cin >> number;  // Step 2: 判断数字的正负性  if (number > 0) {  // Step 3: 如果数字为正，则执行循环  for (int i = 1; i <= number; ++i) {  // Step 4: 在循环中输出数字  std::cout << i << " ";  }  std::cout << "是正数。" << std::endl;  }  else if (number < 0) {  // Step 5: 如果数字为负，则执行另一个循环  while (number < 0) {  // Step 6: 在循环中输出数字  std::cout << number << " ";  ++number;  }  std::cout << "是负数。" << std::endl;  }  else {  // Step 7: 如果数字为零，则输出零  std::cout << "输入的数字是零。" << std::endl;  }  // 询问是否继续  std::cout << "是否继续计算？(y/n): ";  std::cin >> continueFlag;  } while (continueFlag == 'y' || continueFlag == 'Y');  // Step 8: 程序结束  std::cout << "程序结束。" << std::endl;  return 0;  } |   The procedure structure graph is showed in following figure.  ­­­­  Fig. Procedure structure graph for testing case design  Since there are four judging vertexes, the complexity is according to the principle of counting graph complexity.  **Experiment 2.3**  Since the complexity is 5, we can obtain 5 independent paths. Thus, we can design 5 different test cases to cover all the branches and all the statements.   |  |  |  | | --- | --- | --- | | **Case No.** | **x** | **Execution path** | | 1 | 5 | 10 14 15 16 20 15 38 40 14 | | 2 | -3 | 10 14 24 26 27 24 38 40 14 | | 3 | 0 | 10 14 33 38 40 | | 4 | 2.5 | 10 14 | | 5 | -2 | 10 14 24 26 27 24 38 40 |   We use a program to do testing on these four testing cases, the result is showed in following figure.      Fig. Inputs and corresponding execution paths |
| **Conclusion** |
| **Exercise 2.1:**  **In conclusion, our experiment successfully developed a Pothole Tracking and Repair System (PHTRS) using the Vue framework and Spring Boot framework by java, which implements the MVC structure, featuring a user-friendly interface for effortless citizen reporting of pothole details.**  **The system efficiently captures essential pothole information, such as street address, size, location, district, and repair priority, aiding the public works department in strategic planning and execution of pothole repairs.**  **The system has 3 kinds of user Admin, Citizen and Worker.**  **Citizens are the user who report the holes(adding) the holes information. Worker can change the important information of holes including repair crew information, equipment , repair hours, status, material , and costs. They can tell the admin for confirming their work. The Admin takes the responsibilities of confirming the holes already been fixed.**  **After the development of PHTRS, we have a deep understanding of MVC structure. How we code using the MVC structure. Meanwhile, we learn the way using some tools to help us develop a software in the MCV structure and understand how they connect and work together.**  **Exercise 2.2-2.3:**  **Experiment Summary: C++ Program with User Input and Flow Control**  **This experiment involved developing a concise C++ program incorporating user input, conditional statements, loops, and flow control. The program prompts users for an integer, assesses its sign, and executes specific actions accordingly. Users are provided the option to continue or exit after each computation.**  **Observations and Conclusion:**  **Testing covered various execution paths with positive, negative, and zero inputs, demonstrating correct behavior. The addition of a loop for multiple calculations enhanced user interaction. The program's clear conditional statements and loops contribute to readability and maintainability. Overall, this experiment provided practical insights into implementing control flow in C++ programs, offering a foundation for handling user input and executing conditional logic.** |
| **Teacher’s Comments and Score** |
| Comment：  Score：           Signature：                                                 Date： |