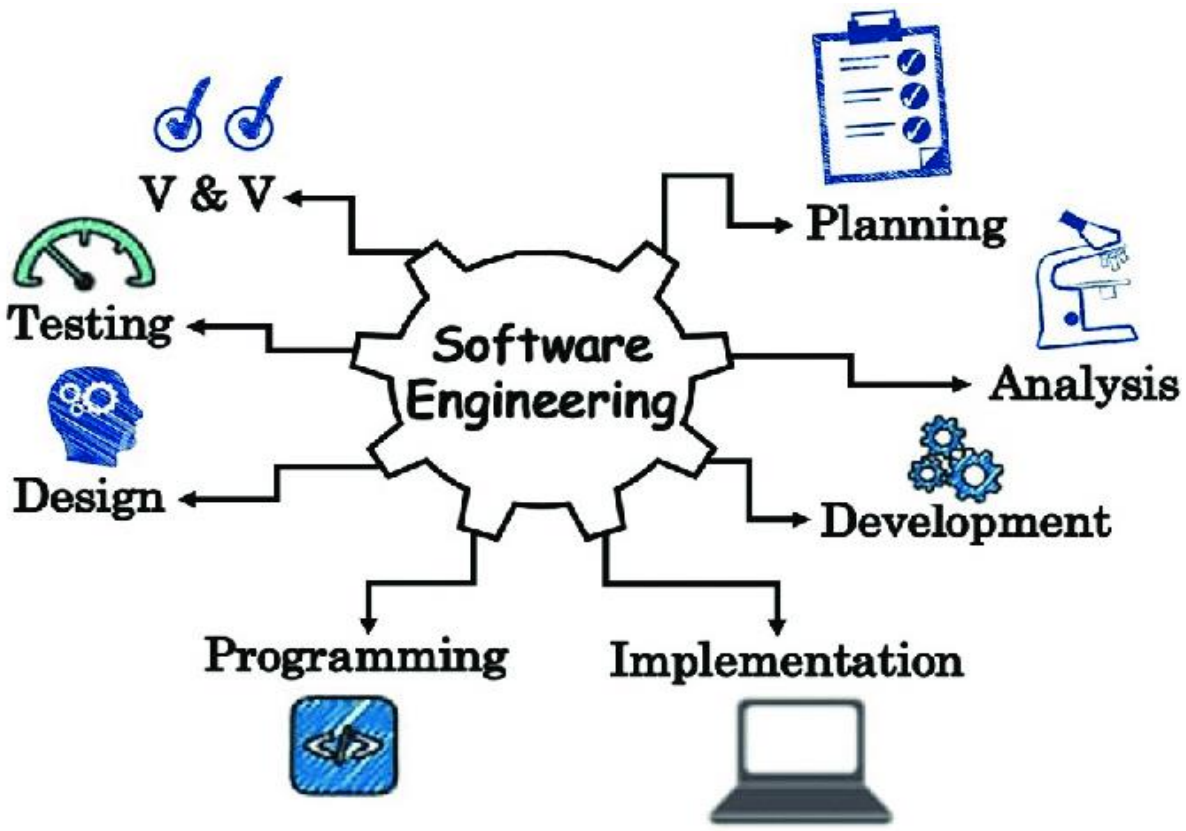

Final Assessment

Introduction To Software Engineering

Sajid Ali (21193046)

5/29/23

Documentation

Introduction

Our task entails developing educational software, with a focus on developing a programme that aids in understanding weather patterns and the effects of weather changes in various regions based on country and month, as well as computing the average temperature of a city. The purpose of our weather programme is to provide people a full introduction to this topic.

To do this, we developed a flexible piece of software that can be accessed via a console interface or a web browser. The application was created using the Java programming language, which is renowned for its dependability and adaptability.

Users may easily access the Weather Software using their prefered web browser by choosing a web-based application. The programme uses localhost on port 8080 to operate locally on the user's computer. To ensure smooth operation, it's crucial to keep in mind that the programme needs a Web or application server, like Apache Tomcat or Glass Fish, to be operating in the background.

In short, the purpose of our educational software project is to offer an engaging environment for learning about the weather and its effects. Users have a wonderful chance to investigate weather-related ideas and deepen their understanding of this dynamic subject with the help of the Weather Software, which is developed in Java and accessible through a web browser.

Modularity

we have identified two modules from the two given scenarios, from the first scenario we will write the module named `find_season`. and from the second scenario we write the module named `find_temperature`.

Module : `find_season(String country, int month)`

- Input Parameter: country
- Input Parameter: month
- output Season

Discription

For this module we are passing the input parameter through keyboard and print on screen as well as on application

The purpose of this method is to determine the season based on the provided country and month.The method starts by retrieving a `seasonsInfo` map, which contains information about seasons for different countries. This map is structured as follows: the keys represent the country names (in lowercase), and the values are lists of maps. Each map within the list contains a season name as the key and a `SeasonDuration` object as the value.

Module : `find_temperature(String city, String time_of_day, int temperature)`

- Input Parameter: city

- Input Parameter: time_of_day
- Input Parameter: temperature
- output Average temperature

Discription

For this module we are passing the input parameter through keyboard and print on screen as well as on application

The method performs a series of checks and calculations to determine the relationship between the input temperature and the average temperature of the given city and time of day. To calculate the average temperature for the specified city and time of day. It iterates over a list of maps containing daytime temperature data for the city and checks if the specified time of day exists as a key in any of the maps. If found, it retrieves the corresponding temperature value and stores it in the `average_temp` variable.

Below is the code snippet for Season Module

```
//  Function to find the season of a country in a particular month
  public String find_season(String country, int month){

      Map<String, List<Map<String, SeasonDuration>>> seasonsInfo =
getSeasonsInfo();

      if(month > 12 || month < 1)
          return "month_invalid";
      if(!seasonsInfo.containsKey(country.toLowerCase()))
          return "country_invalid";
      List<Map<String, SeasonDuration>> seasonMap =
seasonsInfo.get(country.toLowerCase());
      for (int i = 0; i < seasonMap.size(); i++) {

          Map<String, SeasonDuration> stringSeasonDurationMap = seasonMap.get(i);
          Map.Entry<String, SeasonDuration> next =
stringSeasonDurationMap.entrySet().iterator().next();
          int start_month = next.getValue().getArr()[0][0];
          int end_month = next.getValue().getArr()[1][0];
          if(start_month < end_month){
              if(month >= start_month && month <= end_month){
                  return next.getKey();
              }
          }else{
              if(month >= start_month || month <= end_month){
                  return next.getKey();
              }
          }
      }
      return "invalid";
  }
```

Here is how you will access the above function

```
String country = "Pakistan";
int month = 5;
String result = find_season(country,month);
// Displays the result of execution
System.out.println(result); // SUMMER
```

- Find temperature difference

```
 // Finds temperature difference of a city
   public String find_temperature(String city, String time_of_day, int
temperature){
        Map<String, List<Map<String, Integer>>> citiesTemperature =
getCitiesTemperature();
        String message = "";
        // check if city exists
        if(!citiesTemperature.containsKey(city.replace("_"," ").toLowerCase())){
            return "city_invalid";
        }
        if(temperature > 60 || temperature < -20){
            return "temperature_invalid";
        }

        int average_temp = 0;
        List<Map<String, Integer>> daytimeList =
citiesTemperature.get(city.replace("_", " ").toLowerCase());
        for (int i = 0; i < daytimeList.size(); i++) {
            if(daytimeList.get(i).containsKey(time_of_day.toLowerCase())){
                average_temp = daytimeList.get(i).values().iterator().next();
            }
        }

        int diff = temperature - average_temp;

        if(diff > 5) {
            message = "Temperature is more than 5°C above average.";
        } else if(diff < -5) {
            message = "Temperature is more than 5°C below average.";
        } if (diff > 0) {
            return  "Above average " + message;
        } else if (diff < 0) {
            return  "Below average " + message;
        } else {
            return "Equal to average";
        }
    }
```

Here is how you will access the above function

```
String city = "London";
String day_time = "Morning";
String temperature = 19;
String result = find_temperature(city, day_time, temperature);
// Displays the result of execution
System.out.println(result); // Above average Temperature is more than 5°C above average.
```

Tool and Technologies

Following are the technologies we have use in this project:

- Java 17
- Apache Tomcat 9.0
- JSP
- Servlets
- IntelliJ IDEA

Module Descriptions

Implementation in Application

However, it's a web based application. It needs to be deployed at application server or servlet container such as Apache Tomcat in order to function correctly. IntelliJ IDEA comes with a built-in Tomcat's support, upon execution, it auto deploy the WAR (Web Archive) file into the tomcat's deploy folder.

The Application can be further split down into number of small modules, but for the sake of simplicity and better understanding We have focused on following main modules as shown below.

1. **Data**

   Description

- *Name*: Weather Data
- *Objective*: As the name suggest this module is responsible for holding and providing the data, required by the entire web application. Since at this stage our web application aims to serve only two purposes i.e. to find a country's season and to find deviation with an average temperature of a city. Therefore this module currently responsible to provide us SeasonData and CitiesData.
- *Behave*: For the sake of simplicity, I've added a dummy data inside this module. Therefore it's not interacting with any external system (i.e. file or database).
- *Input*: As mentioned, module has hard-coded data values. Hence it does not take data as an input.
- *Output*: Other module can access it, by calling its public function which return data value.

2. **Service**

   Description

- *Name*: Weather App Service

- *Objective*: This module is like an engine of the entire application. This is where the actual implementation exists. It mainly serve two purposes. Finding a country's season and Find temperature difference of a city with its average temperature.
- *Behave*: Module interact with Data module (described above) to perform its action against the input data such as finding a city or a country.
- *Input*: This module takes input as an argument.
- *Output*: After processing, both sub-modules (find_season, find_temp) returns the output as string value.

3. **Controller**

   Description

- *Name*: Weather App Servlet
- *Objective*: Module enhance app's basic functionality. It is in-charge of managing operations and responding to user requests that originate from HTML pages or View Modules. Internally, it transmits data from the View Module to the Service Module and then correctly updates the View.
- *Behave*:

Below is a typical behavior it follows:

```
User =>
        Web Browser (View Module) =>
                    Server (Controller Module) =>
                               Processing (Service) =>
                                        Return View (View Module)
```

- *Input*: Takes input as part of request parameter
- *Output*: Redirect to the View Module.

4. **View Description**

- *Name*: Weather App View
- *Objective*: It is responsible to serve web pages. It allow users to interact with a web interface. Users can select UI elements using cursor to run the application. Moreover, it uses visuals (UI) for an interactive user experience throughout.
- *Behave*: User of this application comes in contact with the module via browser. It further communicate/pass user input to the Controller for processing.
- *Input*: User interact with HTML or JSP elements i.e. (text, drop-down and buttons) through keyboard and mouse.
- *Output*: Return output to the screen as HTML or JSP page.

One can opt manual approach, here are following steps:

- Create a WAR file
  - Go inside the project directory of your project (outside the WEB-INF), then write the following command:

```
    jar -cvf projectname.war \*
```

- If Tomcat is running, stop/kill it.
- Go to the tomcat installation folder (this must be *C:\Documents and Settings\tomcat9x* for you), let's call it <tomcat>.
- In <tomcat>, delete the *temp* and *work* folders. They only contain temporary files.
- If it's a *jar* file maybe is for configuration, so drop it in <tomcat>/lib folder. If it's a *war* file, drop it in <tomcat>/deploy or in <tomcat>/webapps folder.
- Start your tomcat.

How good is Modularity

The Weather App is well organized. Code is written using Java programming language. Application is divide into various packages, where each package serves a specific purpose. Following are the packages in this app:

- Data: Hold/Server data to the app.
- DTO: Data Structure.
- Service: Business logic for find_temp and find_season.
- Servlet/Controller: communicate between Service and Web Module.
- Web/HTML pages: HTML page and views are defined here.
- Test: It contains all the test cases.

Here is the Application's directory in tree structure.

```
—src
    ├───main
    │  ├───java
    │  │  └───com
    │  │      └───learning
    │  │          └───weatherapp
    │  │                  ├───data
    │  │                  ├───dto
    │  │                  ├───service
    │  │                  └───servlet
    │  ├───resources
    │  └───webapp
    │          ├───images
    │          └───WEB-INF
    └───test
          ├───java
          │    └───com
          │        └───learning
          │            └───weatherapp
          |───resources
```

Review Checklist

- Read all the requirements and understand them well.
- Prepare a design (enough to meet the application's basic goals).
- Think about what modules you may need in this application.

- Document them all.
- Brainstorm to identify suitable branches names for your version control.
- Start writing code.
- Initialize a Git directory and add the basic app to branch named'basic-version'
- Enhance your code to make it more user friendly and add the updated code to a new branch named 'complete-version'
- Write test cases to see if everything works or behave as intended.
    - At this stage you can perform rigorous testing on your application to see how well it perform.
    - Test with valid and invalid inputs.
- If you find any ambiguity fix those issues and repeat the tests again.
- If everything works fine, Deploy the app.

Black-Box Test Cases

A black box test can simulate user activity and see if the system delivers on its promises. In this application we have covered following:

- **EQUIVALENCE PARTITIONING**

In this type of testing we divide the input domain into classes of data (valid and invalid range). Here we will take two invalid and one valid equivalence classes.

Assumption: We are providing a valid country name

Equivalence Partitioning (Season)

Invalid	Valid	Invalid
month < 1	1-12	month > 12

Equivalence Partitioning (Temperature)

Invalid	Valid	Invalid
temperature < -20	-20-60	temperature > 60

- **BOUNDARY VALUE**

Boundary Value Analysis (Season)

Invalid	Valid	Invalid
month = 0	1 – 12	month = 13

Boundary Value Analysis (Temperature)

Invalid	Valid	Invalid
temperature = -21	-20 – 60	Temperature = 61

White-Box Test Cases

One can test the quality or internal design of code. Let's say we perform white box test to ensure system return the correct records that we have for demonstration purpose we will check the season data size in our system.

Test Implementation and Execution

Black Box Test Implementation

Equivalence Partitioning: Below program demonstrate the equivalence partitioning

INVALID: month value below 1

```
Assertions.assertEquals("month_invalid",seasonService.find_season("Pakistan",-5));
```

VALID: month value between 1-12

```
Assertions.assertNotEquals("month_invalid",seasonService.find_season("Pakistan",7)
);
```

Boundary value analysis: Below program demonstrate the equivalence partitioning

INVALID: month value above 12

```
Assertions.assertEquals("month_invalid",seasonService.find_season("Pakistan",14));
```

INVALID: month shouldn’t exceed 12 or can't be below 1

```
Assertions.assertEquals("month_invalid",seasonService.find_season("Pakistan",13));
Assertions.assertEquals("month_invalid",seasonService.find_season("Pakistan",0));
```

VALID: month should be between 1 and 12

```
Assertions.assertNotEquals("month_invalid",seasonService.find_season("Pakistan",
1));
Assertions.assertNotEquals("month_invalid",seasonService.find_season("Pakistan",
12));
```

White Box Test Case

```
final int SEASONS_DATA_SIZE = 4;
Assertions.assertEquals(SEASONS_DATA_SIZE, seasonService.getSeasonsInfo().size());
```

Version Control

We first initialized Git, which automatically creates the main branch when we initially push the code to GitHub. After that, we set up the folder structure for the code and documents, along with a brief readme file. To have a complete version of the code for production, we need to create another branch where we will push the final code. Additionally, we create a branch called "boilerplate-code" to establish the application layout. This branch contains the folder and file structure of the application.

Application has following three branches

- main (production-ready-version, contains test case, documentation etc.)
- boilerplate-code (hold all the fundamental application folder structure)
- complete-version (improved version. Better user experience through JSP/HTML pages)

Ethics

Different negative outcomes might result from a lack of ethics and professionalism in the design and implementation of code. Here are a few illustrations:

- Privacy breaches: Developers may create code that collects and shares sensitive user data without the appropriate consent or security protections if they do not prioritise ethical privacy issues. Unauthorised access to personal data might occur from this, which could cause fraud, identity theft, or other privacy breaches.
- Security vulnerabilities: Inadequate data validation or insufficient authentication techniques are examples of security flaws that might arise from unprofessional coding practises. With the potential to result in financial loss, harm to one's reputation, or even bodily harm in vital systems, hackers can use these vulnerabilities to get into systems, steal sensitive data, or disrupt services.
- Algorithmic bias: Biassed results may result from designing algorithms without taking ethics into account. For instance, in the criminal justice system, algorithms may unfairly target particular ethnic or socioeconomic groups if they are trained on biassed data, reinforcing systemic inequalities.

Additionally, Dependence on the system could have negative effects if the system is fed with inaccurate data. As a result, it is advised to conduct a little additional investigation before accepting the results it produced. so, these moral dilemmas should be given careful consideration, taking into account how the programme affects people, communities, and the environment. To guarantee the adoption of ethical and inclusive practises, it is important to regularly assess and look into the project's ethical implications.

Professionalism

Here are two recommendations based on the ACS (Australian Computer Society) or IEEE-CS (Institute of Electrical and Electronics Engineers Computer Society) ethical rules to help you keep your suggested programme morally and professionally correct:

- Ensure Privacy and Data Protection: Put in place strong safeguards to secure sensitive data inside your programme and protect user privacy. abide by the concept of informed consent, which requires that users be made fully aware of the data being collected and the purposes for which it will be used before giving their express approval. To guarantee compliance and safeguard the confidentiality of user information, you should routinely evaluate and review your data handling practises. You should also abide by all applicable privacy laws and regulations.

- Foster Inclusive Design: utilise inclusive design principles to make sure that a wide range of people may access and utilise your programme. Think about the requirements of people with impairments, various linguistic tastes, or various cultural backgrounds. Provide other input methods, allow for various screen readers, and provide simple, user-friendly user interfaces. Test your software frequently with a varied user base to spot any potential biases or obstacles that could exclude particular people or groups and remove them.

Discussion

Let's talk about the Weather Application and consider how we may make it even better. We can concentrate on the following areas:

- User Interface (UI): Examine the process and user interface in use to find any possible usability problems or areas for development. To improve the user-friendliness and intuitiveness of the programme, think about doing user testing or gathering feedback. Since it is now console-based, the end user is not very interested in it.
- Data Validity and origin: Continuously evaluate the accuracy and reliability of the temperature data sources used in the application. Explore options to incorporate multiple reliable data sources or meteorological APIs to ensure up-to-date and accurate information.
- Error handling and confirmation: Implementing strong error messages and validation techniques will improve the application's ability to handle errors. Inform users of any input mistakes so they may provide accurate information.
- Client Feedback: actively seek consumer input and take it into account while developing new products. Assess user needs, preferences, and developing requirements often to set priorities and organise upcoming upgrades.

Conculsion

We have reached a decision after in-depth discussion of the Weather Application. The application's two major purposes are, as was already noted, identifying a country's season for a certain month and calculating the temperature difference from a city's mean temperature. Incorporating the most efficient industry practises has been our goal throughout the whole Software Engineering process, from software design to deployment. We started the project with a simple layout and added more functionality over time. Our focus was on modularity, using a divide and conquer strategy, and ensuring that the issues were separated. These initiatives demonstrate our commitment to using the best software engineering practises. We have created a trustworthy and user-friendly application that accurately gives season data and temperature variations by adhering to professional standards, ethical concerns, and continually developing our design and functionality with a a user-freindly web-based application. We are pleased with the results of this project and think it will successfully accomplish its goals.