Dalton Rose

CS-320

2024/04/23

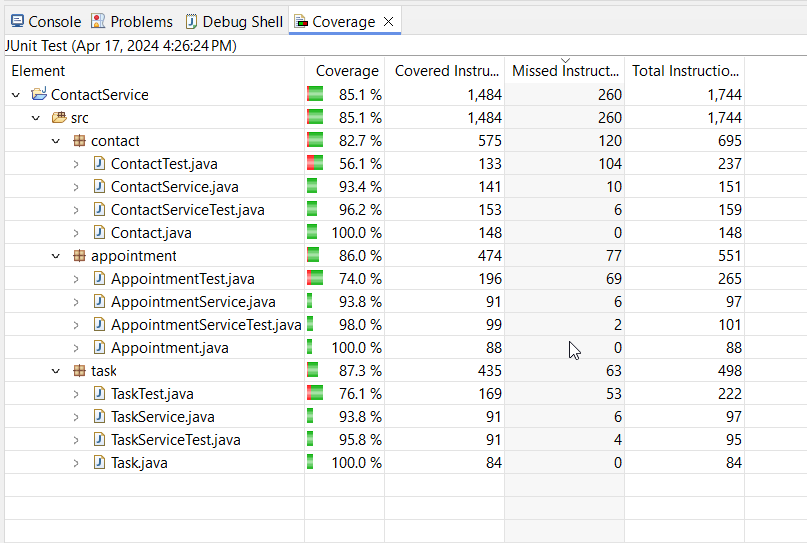
**Summary and Reflections Report**

When designing unit tests for this software, the software requirements were very important in establishing exactly what things I needed to test. In order to test that my software met the software requirements, I broke down the requirements into smaller pieces and tested individual aspects of the requirements. For example, the Task class required that the task ID must not be null and that the string must be less than 10 characters. Based on these requirements, I designed two unit tests: one for creating a task with a null task ID and another for creating a task with a task ID greater than the 10-character limit. Each of these scenarios should result in an illegal argument exception being thrown. I utilized this same process in designing unit tests for the other elements of the software. By directly testing against scenarios laid out in the software specifications, I am confident the software behaves as expected.

A close up of text

Description automatically generatedA close-up of a computer screen

Description automatically generated

JUnit coverage was a valuable tool in ensuring that all aspects of the software were tested. My overall JUnit coverage was 85.1% and if you exclude the test files, my coverage reaches an average of 96.8%. These coverage values indicate that almost all aspects of the code are being tested. In the few cases where 100% coverage is not reached, the only aspects that were not covered were often scenarios where there is currently no functionality, such as updating the name of a non-existent task. Overall, I’m confident based on the results of the JUnit tests and their coverage that my program performs as intended.

To ensure that my code was efficient and technically sound, I designed my unit tests to only test one specific aspect of the program. Proper care was taken to ensure that my code was easy to read and utilized descriptive names and comments. In my unit tests, I utilized display names to better describe the tests in a more readable way. I made sure to utilize variables for many of the tested inputs in the unit tests so that tests could be updated quickly if the software requirements were to change.

A close-up of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

For this program, I utilized JUnit to perform unit testing to verify the functionality of the program and that the program met the software requirements. Unit testing is a form of software testing that focuses on testing isolated portions of code. Therefore, in these JUnit tests, I primarily focused on testing the behavior of each individual function to ensure that it functioned as intended. For example, I designed JUnit tests for the Task class that tested functions to create tasks, update the information associated with a task, and retrieve information about a specific task. These tests also included testing for the use of parameters outside of the program specifications, such as a ID or name that is too long.

Unit testing is only a very small part of thoroughly testing software and there are a variety of software testing techniques that were not utilized to test this program. One such technique is integration testing, which tests that the individual components of the software can communicate and interact as intended. As my program had no meaningful interaction between the various components (with the exception of the service classes to manage tasks, appointments, and contacts), I was not able to incorporate thorough integration testing. Integration testing for this program might involve testing whether appointments can be successfully assigned to a specific contact. Another type of testing that I haven’t utilized is system testing, which tests the functionality of the overall system. System testing for this program might involve testing that you can access, add, and modify things such as contacts, tasks, and appointments using an interface. Additional testing techniques that were not used include performance testing, compatibility testing, and security testing. These testing techniques are important to ensure that data is secure, the software is fast and responsive, and the software functions correctly across all supported platforms. To properly ensure the quality of the software, it would be necessary to employ many testing techniques in addition to unit testing.

When designing and developing this software, I employed caution by making sure that the code could be broken down into small pieces. Each function was designed and implemented to only complete a single action. As a result, the unit testing was easier to design and implement. If functions perform a myriad of tasks, it can be much more difficult to test them and determine exactly what is happening when the function is called.

Bias is a valid concern when testing software. As the developer of the program, I deeply understand all elements of the code and my intentions when writing the code. If a third party were developing tests for the program, they might not know what needs to be tested for until they gain a better understanding of the codebase. Furthermore, if I’m not cautious, I might not test my code thoroughly enough because I assume it functions exactly as I intended and not how it was written. Therefore, it’s important to make sure your tests are thorough to eliminate the possibility of bias affecting the quality of the software.

In software development, it is very important to practice discipline. Practicing discipline in software development can greatly improve the quality and maintainability of the software. Cutting corners now could also cost the development team a lot of time and money later. Furthermore, cutting corners and not doing due diligence could have serious consequences for the company, the users, or other parties. For example, data breaches present a serious risk but have unfortunately become a relatively common occurrence. However, data breaches are not the extent of the consequences of becoming complacent in development and testing. In some cases, like medical equipment software, not being thorough during software development and testing could result in deaths. For these reasons, it’s extremely important to employ proper software development and testing techniques and practice discipline during software development.