Richard Chang

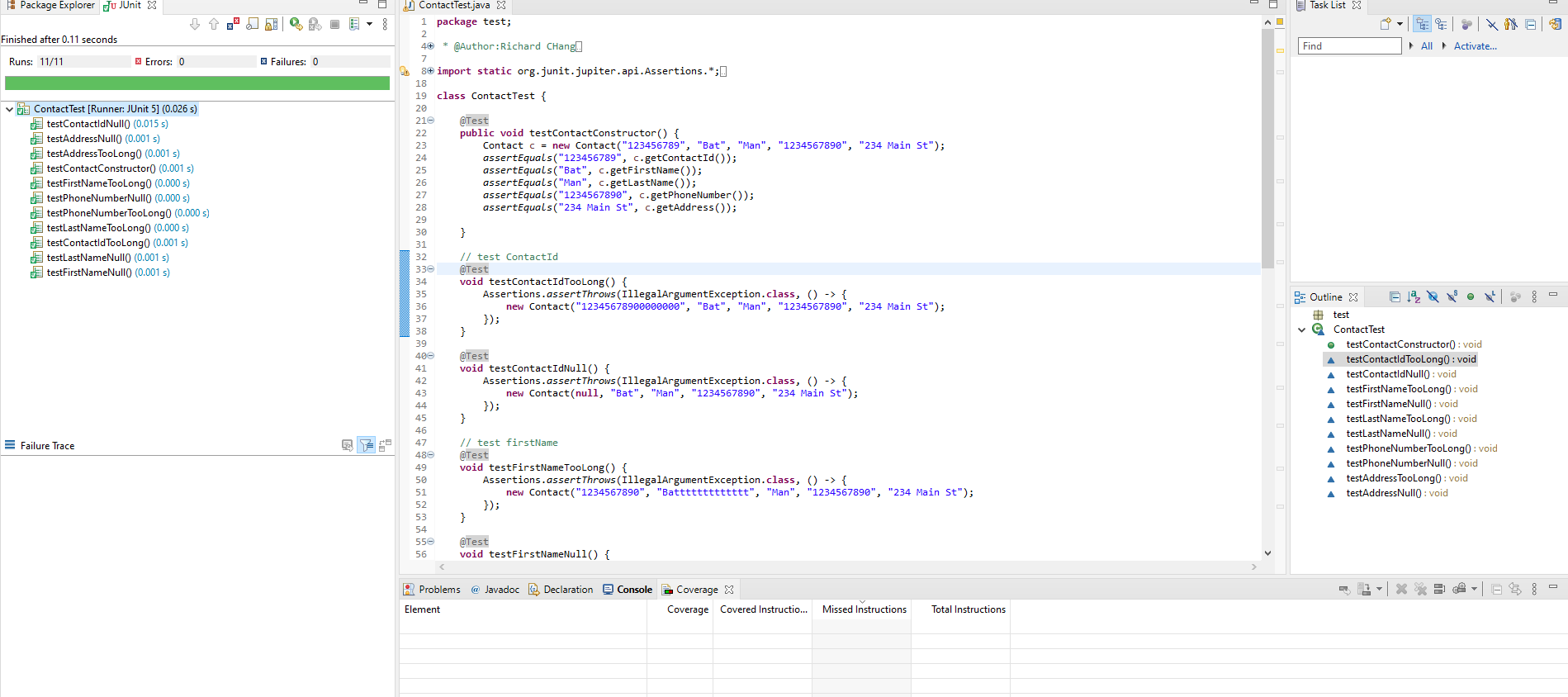
Professor Wraithe

CS320

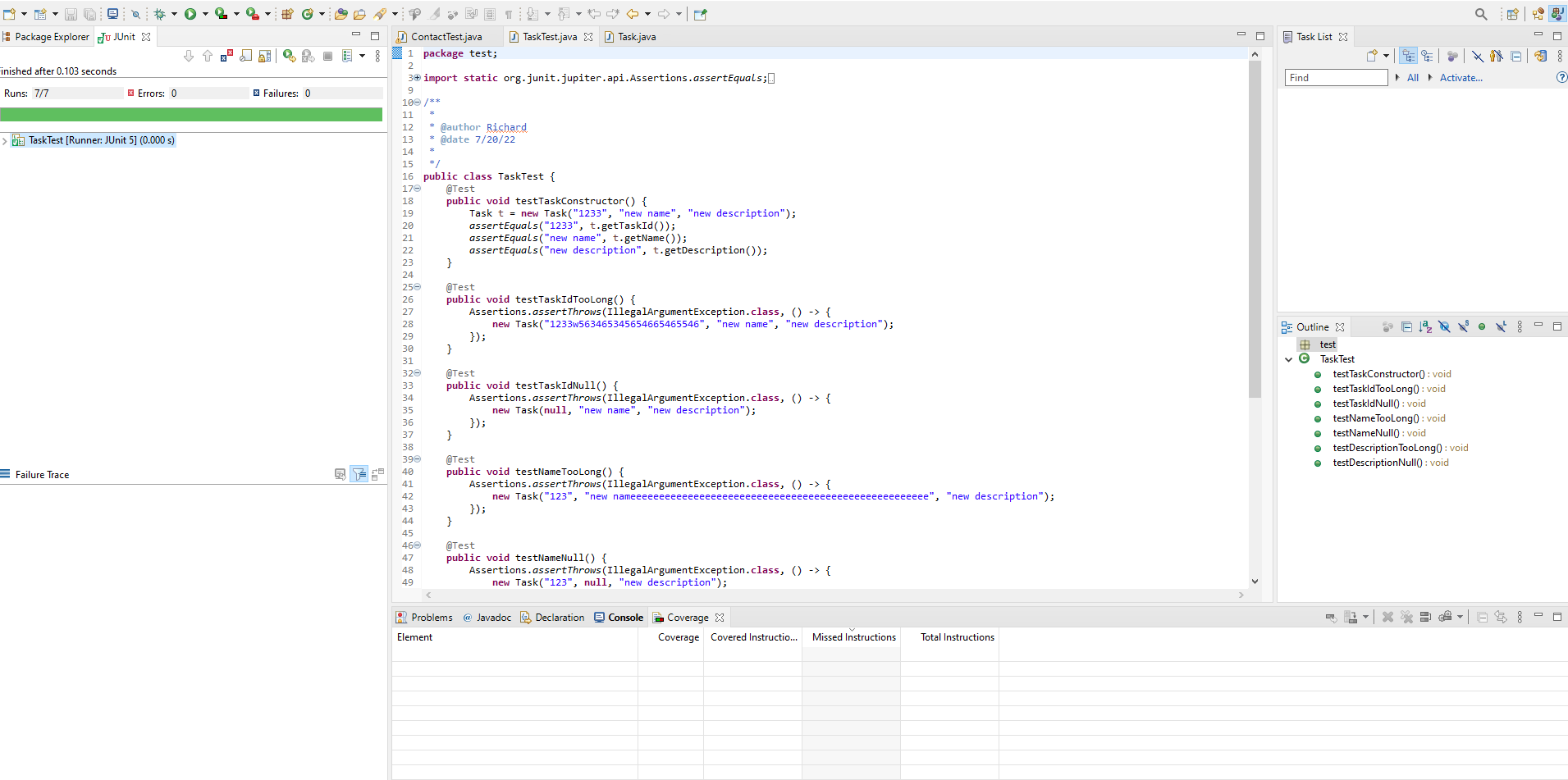
August 10, 2022

Project Reflection

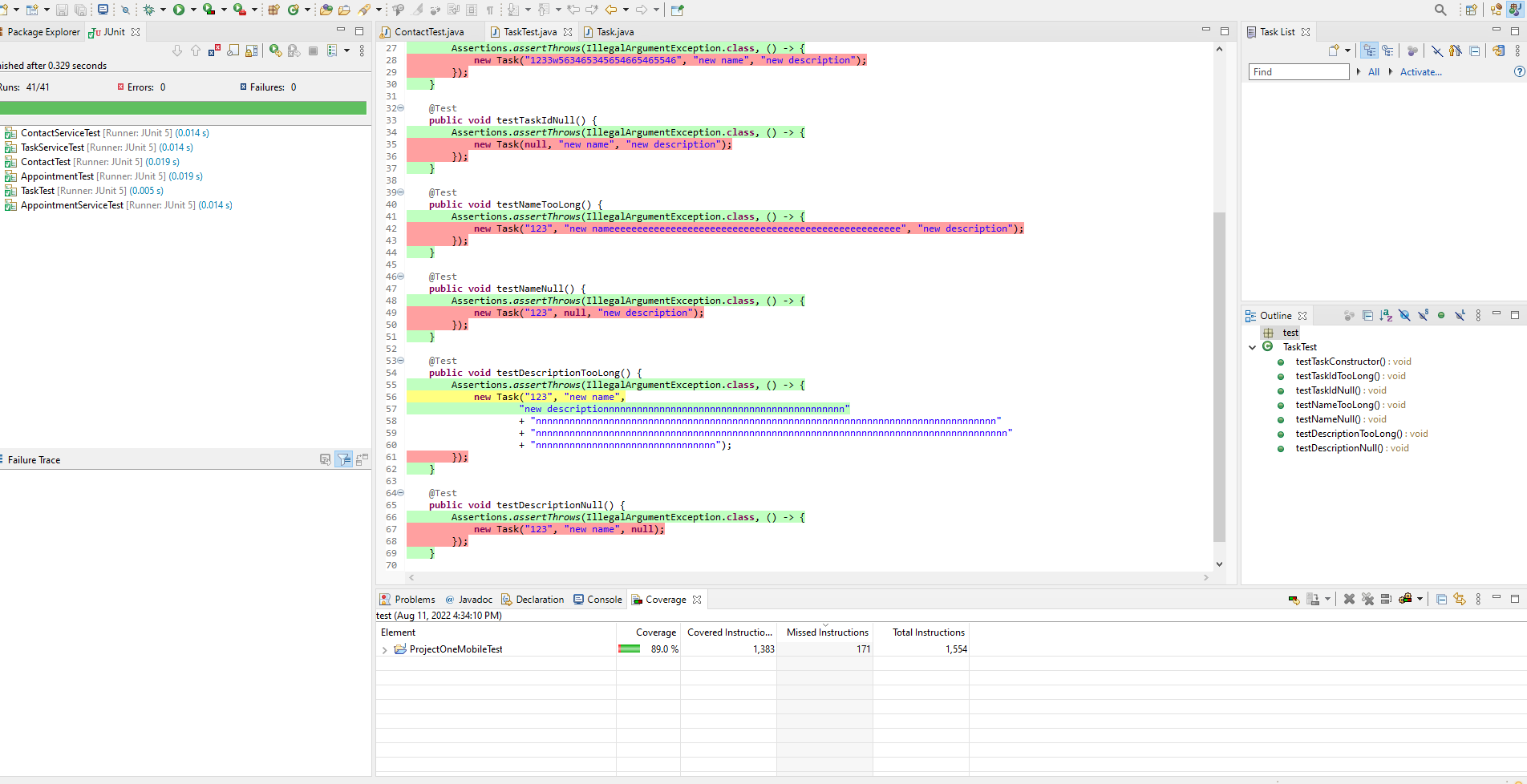
Software requirements are defined as the agreed statements of product features. Key features are defined by the product owner; each team member of the development team ultimately decide what is actually built in small story chunks. In an Agile process, each sprint may have a product demo and test stage, such as JUnit testing, which will provide feedback to the team’s position on product development. Fixing and refining the product often leads to the most high quality and comprehensive product. In requirement determination, feedback from the client greatly increases the business value in the product. Following such protocol, many ways instruct testing: unit testing, integration testing, system testing, and user acceptance testing. Unit testing is the step to find errors or bugs in small functions, code components, or small features and one of the most popular acts of code verification. I will go over software requirements. In the examples below, the contact JUnit test provides some insights on the checks on the Contact class. In the Contact class, requirements detail that a constructor has 5 parameters. The contactId must be unique and not more than 10 characters and not null. The firstName must be not more than 10 characters and not null. The lastName must not be more than 10 characters and not null. The phone number must be 10 characters and not null. The address must be not more than 30 characters and not null.

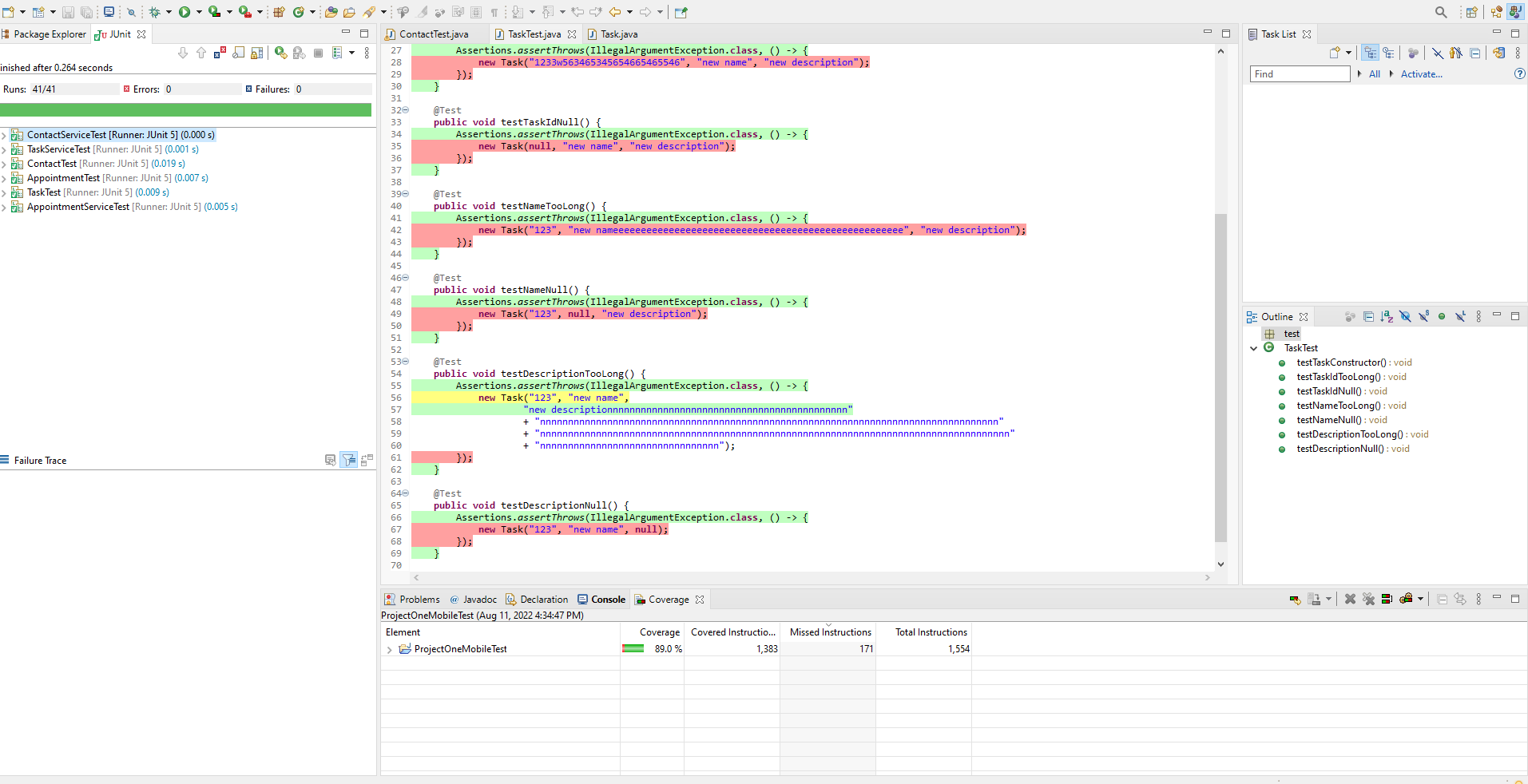


For the task class, requirements detail that a constructor has 3 parameters. The taskId must be unique and not more than 10 characters. The name is not more than 20 characters and not null. The description is not more than 50 characters and not null. As you can see below, the tests pass which indicate a confirmation of a good fulfillment in software requirements.



As the Junit cases proceed, another great indicator is code coverage. Code coverage checks the overall source code against the test cases. The recommended code coverage threshold is 80% (Pittet, n.d.). I can check for code coverage by clicking code coverage on the test package or the whole project. In both cases, as shown below, the code coverage indicates 89.0% code coverage which is greater than 80% and assumes satisfactory completion of Junit tests. The degree of most project in the real world aim for 100% but testing takes time, effort, and resource (Dean, 2019). Optimizing source code also requires optimizing the sanity of making tests.





Technically sound code is code without syntax errors and gives a summary of what the lines of code do. For example, if I want to code an exception is the appointment id is null or greater than 10 characters, then this is the code,

if (appointmentId == null || appointmentId.length() > 10) {

throw new IllegalArgumentException("Invalid appointment id - null or length > 10");

}

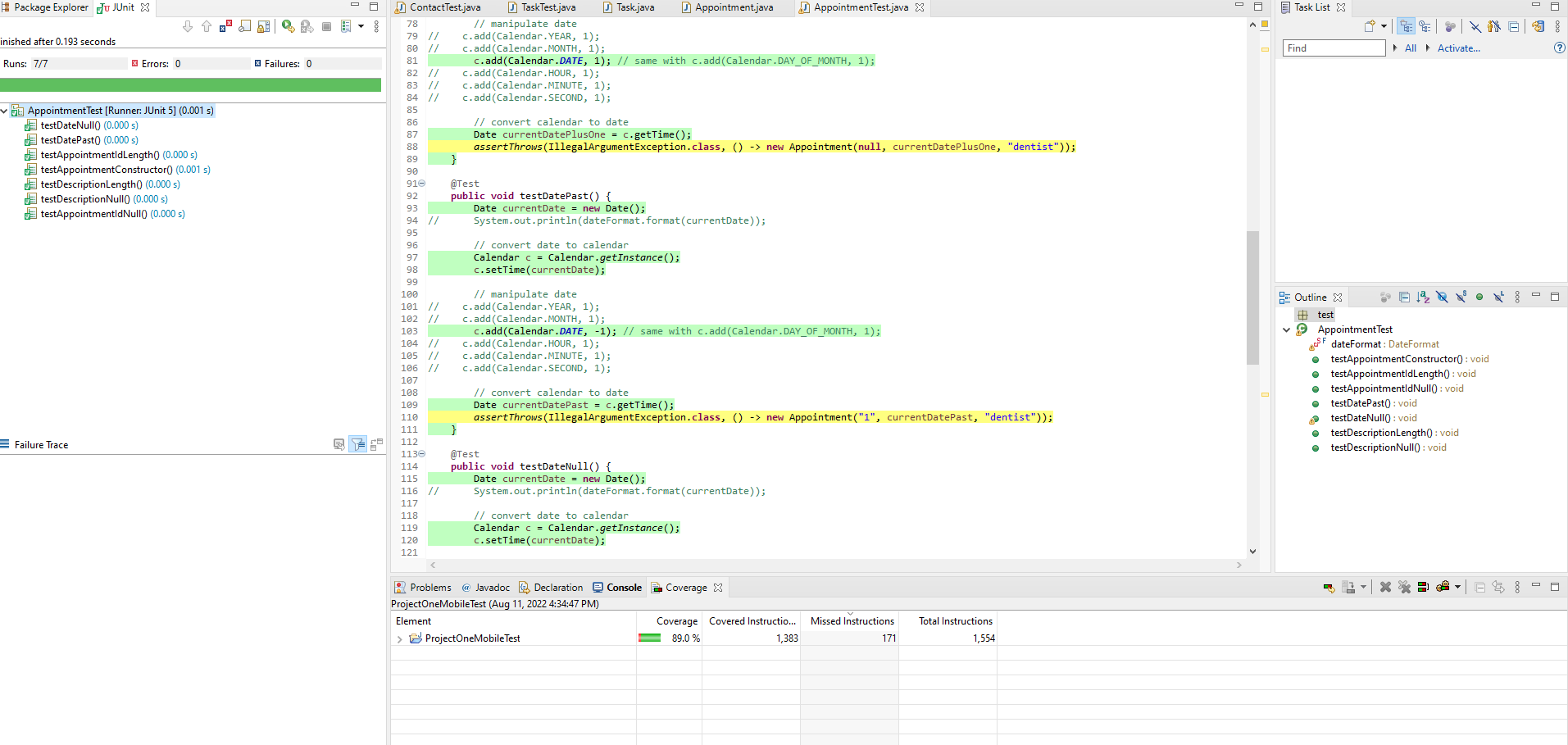
If I want to code an exception for the description of the appointment, then this is the code:

if (description == null || description.length() > 50) {

throw new IllegalArgumentException("Invalid description - null or length > 50");

}

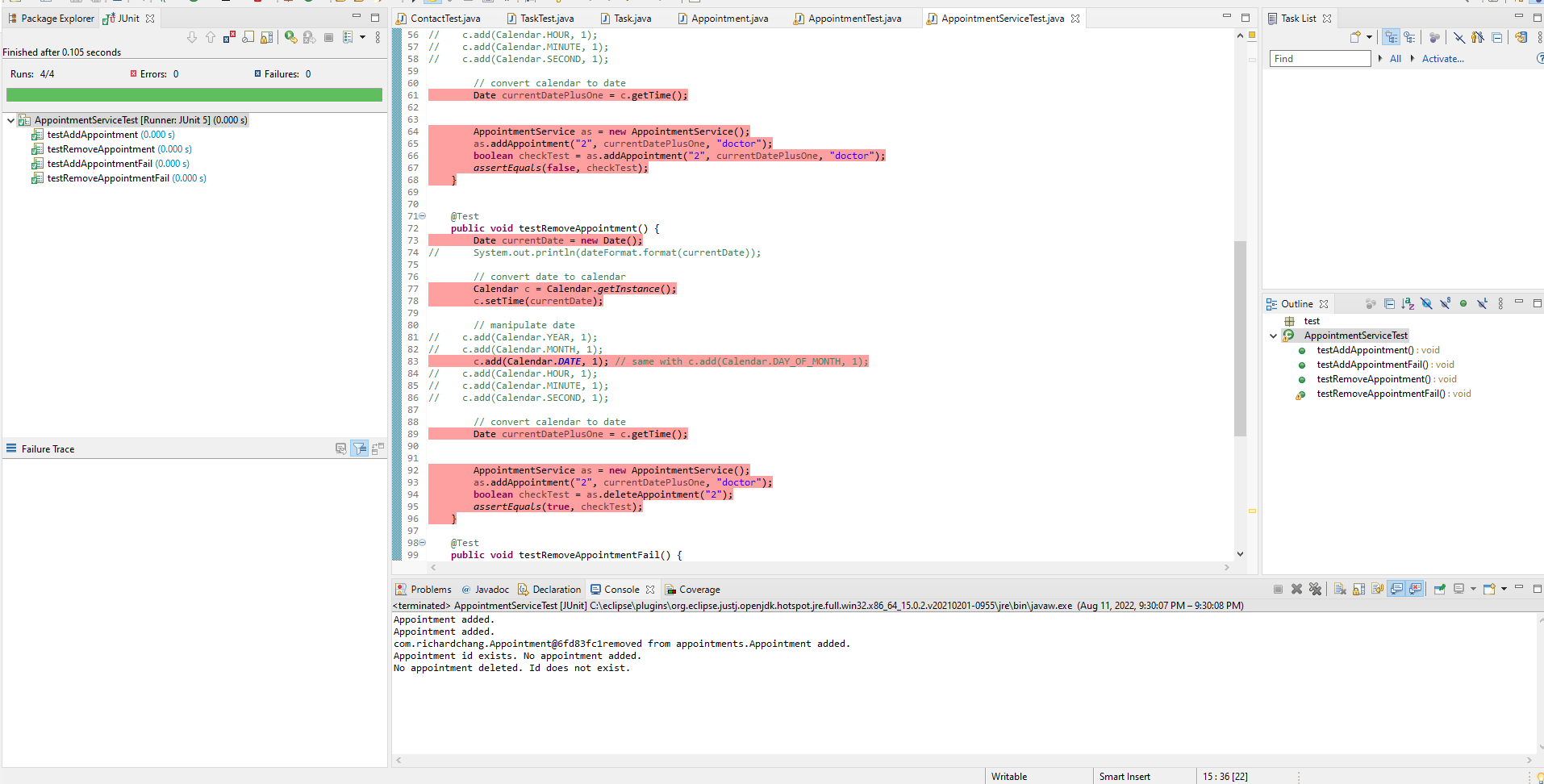
Under the Appointment Test, another Java class JUnit test, I see that the test cases pass and the code is technically sound to me and the machine. See below screenshot.



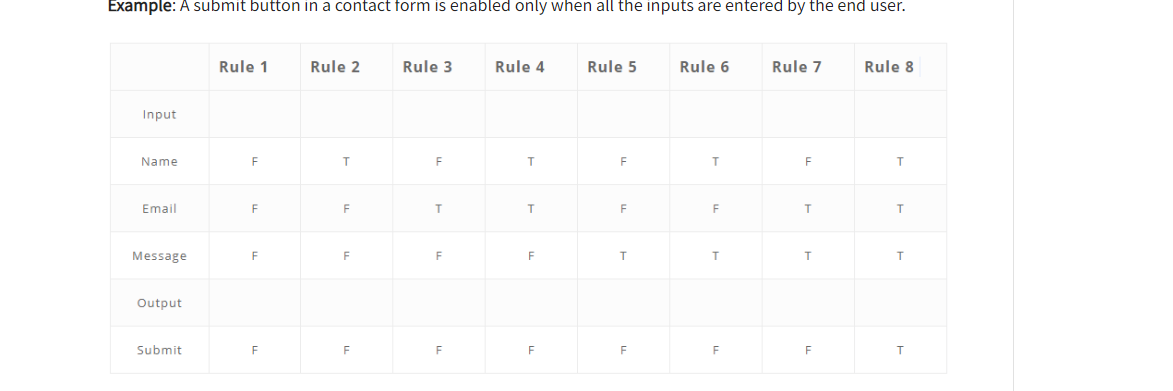
Code coverage is one of the factors of efficiency. Each line cross references with the source code to validate the performance of code. 83.0% in the AppointmentTest Junit test is an acceptable response for code efficiency.



Software testing techniques include boundary value analysis, equivalent class partition, decision table in state transition changes, and error guessing. Most of the deployable unit testing is Junit testing which is considered error guessing. Error guessing lists the possible errors or possible error-prone situations (Hamilton, 2022). In the testAppointmentService class, unit testing predicts exception errors when you add zero or more of the appointments and delete zero or more appointments. See below.



The other decision table based testing depicts a table that follows many rules and a single result for each case as shown below (Hamilton, 2022). If you are familiar with decision trees, then each path has a single result when a yes-no answer confirms each step. But in this table, true and false categorizes the “path” for each submit case answer.



Each of the software techniques have merits. In boundary value analysis, extreme test cases follow a series of test that would match mathematically possibilities, such as testing temperatures at extremes in a program. In equivalent class partitioning, each case passes conditions based on similar parameters. In decision table software testing technique, you follow many paths in a decision tree, such as steps to cook a brownie. In state transition technique, the input conditions will change the state, such as setting different PINs a number of times. Finally, error guessing is a fundamental testing technique that guess errors in the software applications. Each software testing technique has benefits and applies unique methodology for consideration.

While working on the project, I did employ caution to the extent that I questioned myself if I covered the application with enough test cases. Sometimes, I corrected method implementation to produce a test case result. For example, in order to add an appointment, my test case used a Boolean in order to return a final result in a Junit test. I was curious about whether I could write the source code in a better manner or cleaner code. I realized some mistakes such as, 10 instead of 20. A possible error may be that the appointment list did not update with one more appointment when added. Code testing often complicates as more and more lines of codes add. As more relationships add, each test case fill with more and more testing situations. The complexity fuels software tester to efficiently build test cases. Bias is limited due to the actual program or Junit testing which runs on the computer. I can have many developers who can provide more test cases and less bias, as a solution. Whether a bug is found in the beginning or end of testing, product’s success builds on the discussion of possible incoming issues. According to Hales, software testing begins from the start and ends at the absence of a product (Hales). A discipline software engineer will not cut corners because issues will occur and the product’s quality dramatically decreases. In order to avoid technical debt, it is best to arm myself with programmer’s knowledge and learn more about aspects of programming. Getting others involved adds to product’s quality. Software testing is a major step to solve a program’s issue before it occurs and enhances the prospects of the best version of a software product.

References

Dean. (2019, December 27). Everything you need to know about code coverage. Codegrip. https://www.codegrip.tech/productivity/everything-you-need-to-know-about-code-coverage/

Hales. (2022, January 31). When does software testing begin in SDLC: Know your testing phases. Medium. Retrieved June 30, 2022, from https://medium.com/software-testing-pipeline/when-software-testing-begins-in-the-sdlc-knowing-your-testing-phases-faf742c2aa39

‌Hamilton, Thomas. Software Testing Techniques with Test Case Design Examples. (2019, December 4). Guru99.com. https://www.guru99.com/software-testing-techniques.html

How can I become technically sound and better in programming? (n.d.). Quora. Retrieved July 21, 2022, from https://www.quora.com/How-can-I-become-technically-sound-and-better-in-programming

IBM. (2019). What is software testing? Ibm.com. <https://www.ibm.com/topics/software-testing>

‌Importance of SDLC in software testing, system and software development life cycle- Performance Lab. A Software Testing Company. (2021, August 17). Retrieved June 29, 2022, from https://performancelabus.com/software-testing-importance-sdlc/

Pittet, S. (n.d.). Introduction to Code Coverage. Atlassian. https://www.atlassian.com/continuous-delivery/software-testing/code-coverage

SDLC - Overview. Tutorials Point. (n.d.). Retrieved June 29, 2022, from https://www.tutorialspoint.com/sdlc/sdlc\_overview.htm

‌