

**Software Development**

**Guided Exercise 3 Writeup**

Section C9.84.19637-1

Megana Atluri and Sarah Hoover

100504469 and 100504446

March 31, 2023

**Table of Contents**

[1. Introduction](#_c5pn64apsti6) 3

[2. Function 1: Equivalence Classes and Boundary Values](#_fu6h91wuaahm) 4

[3. Function 2: Syntax Testing](#_a2amiltrlet7) 5

[4. Function 3: Structural Testing](#_srzo9xpslvne) 6

[5. Conclusion](#_k4t8zalo800q) 8

# Introduction

The purpose of this assignment was to practice test-driven development and different processes to testing. There were three parts to this project, each one focusing on a different way of testing software: equivalence classes and boundary values, syntax testing, and structural testing. The first two are black-box testing approaches and the third is a white-box testing approach. Black-box testing tests the inputs and outputs of the functions, not focusing on the code structure of the method itself, while white-box testing focuses on the structure of the code, ensuring that all possible paths are taken during testing.

There were three functions that it was necessary to implement. The first function requests the shipment of a product, testing using equivalence classes and boundary values. The second function sends a product, testing using syntax analysis. The third function delivers the product, testing using structural testing.

The program was written using PyCharm, and the code for it can be found on Github.

# Function 1: Equivalence Classes and Boundary Values

Equivalence classes and boundary values that we identified

The product id is a valid ean13 code. For the product id, we identified three equivalence classes: one where the input is valid, one where the input is invalid because it is not a number, and one that is invalid because the check sum of the product id failed. There were also two boundary values that were tested for the product id: one where the product id is 12 digits long and the other where the product id is 14 digits long.

The order type attribute describes whether the product is being shipped as a premium product or a regular product. We only had equivalence class tests for this attribute. We had two valid equivalence class tests (one for when the product is a premium product and one for when the product is a regular product). We also had two invalid equivalence classes: one for when the product had an incorrect order type and one for when the order type is not a string.

The next attribute that was tested was the address. We tested both equivalence classes and boundary values for this attribute. The boundary values checked the length of the string and the equivalence classes confirmed that it was a string.

Next we tested the phone number attribute, in which we tested the length with boundary values and the type with equivalence classes.

Then we tested the zip code attribute. We tested the value of the zip code, as not every 5 digit code is a valid zip code, using boundary values. We also used equivalence classes to confirm the length and type of the zip code.

Finally we tested the outputs of the test.

We were sent group 7’s excel file and responded with the following: ​​

“Your excel file looks great! We only noticed two small things. First, tests 5 and 27 might be redundant but we are not sure about that. Second, I am not sure if it makes sense to include test 48 in the excel file because it is not a complete test (again we are unsure).”

Their excel file was complete and thorough and timely (recieved on March 17).

# Function 2: Syntax Testing

## Grammar:

File ::= Start\_object Data End\_object

Start\_object ::= {

End\_object ::= }

Data ::= Field1 Separator Field2

Field1 ::= Data\_label1 Equality Data\_value1

Field1 ::= Data\_label2 Equality Data\_value2

Separator ::= ,

Data\_label1 ::= Quotes Tag\_value1 Quotes

Data\_value1 ::= Quotes Value1 Quotes

Data\_label2 ::= Quotes Tag\_value2 Quotes

Data\_value2 ::= Quotes Value2 Quotes

Equality ::= :

Quotes ::= “

Tag\_value1 ::= OrderID

Value1 ::= a|b|c|d|e|f|0|1…|9| {32}

Tag\_value2 ::= ContactEmail

Value2 ::= Start Email\_symbol Domain Email\_separator Extension

Start ::= a..z0..9

Email\_symbol ::= @

Domain ::= a..z

Email\_separator ::= .

Extension = com

## Derivation Tree:

The derivation tree is located in the docs folder as an svg called Derivation Tree.svg.

We had a total of 51 nodes in our derivation tree. We developed tests by duplicating, deleting, and modifying the nodes, ensuring not to create repetitive tests, but also to only allow modification testing on terminals, not non-terminals.

# Function 3: Structural Testing

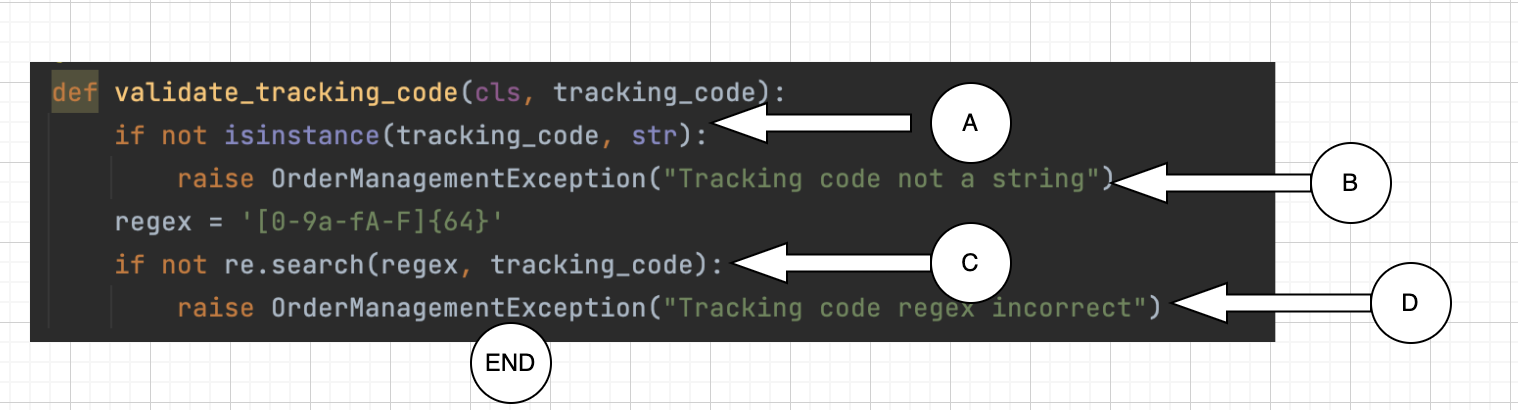
### Control Flow Diagram

The control flow diagram for the deliver\_product method is located in the docs folder named deliver\_order\_control\_flow.drawio.svg. The control flow diagram for the validate\_tracking\_code method (which is called within deliver\_order) is also in the docs folder named validate\_tracking\_code\_control\_flow.drawio.svg.

For the control flow for deliver\_order, there were a total of 14 nodes and 18 edges. This leads to a Mccabe (cyclomatic) complexity of E – N + 2 or 18 – 14 + 2. Which is a total of 6 outcomes. Ergo, 6 tests. These tests are as follows:

1. ACEFGHJLN
2. ABN
3. ACDN
4. ACEFGHIN
5. ABCEFGHJKN
6. ACEFGFMN

Below is a representation of the nodes on top of the code itself.





# Conclusion

In conclusion, we used three types of software testing methods in order to test three different software methods. These three methods were implemented in python and the functions were all related to a shipping company. The first method requested the shipment of a product, the second sent the product, and the third delivered the product.

All three of these methods were tested in different manners and using different testing strategies. The first method was tested using the black-box approach of equivalence classes and boundary value testing, which focuses on grouping cases together based on their expected output and how different inputs will be treated as like. Additionally, this focuses on testing the extrema (boundaries) between classes. The second method was tested using syntax testing, which uses a grammar to parse the expected input and tests the method based on the duplication, deletion, and modification (in the case of terminals) in order to test the method. The third method was tested using structural testing. In structural testing, a control flow diagram is created based on the structure of the code and paths are determined from this diagram. These routes are the different paths that can be executed through the code while it is executing. Each of these paths must be tested in order to ensure that all possible routes the execution can take are tested.