Basic Unit Testing Theory

Unit testing

- Use assertions to determine whether method outputs are correct given a set of inputs
- Unit tests should run in isolation from one another
- Unit tests should be built incrementally, or before we write any functionality
  - TDD

Why do we unit test?

- Identify defects early in the development cycle
- Small bugs can lead to chaotic system behavior
- Testing impacts the design of your code
- Testing forces you to slow down, read your own code, debate against yourself
- Automated tests (continuous integration) support maintainability and extensibility

Common reasons people don’t unit test

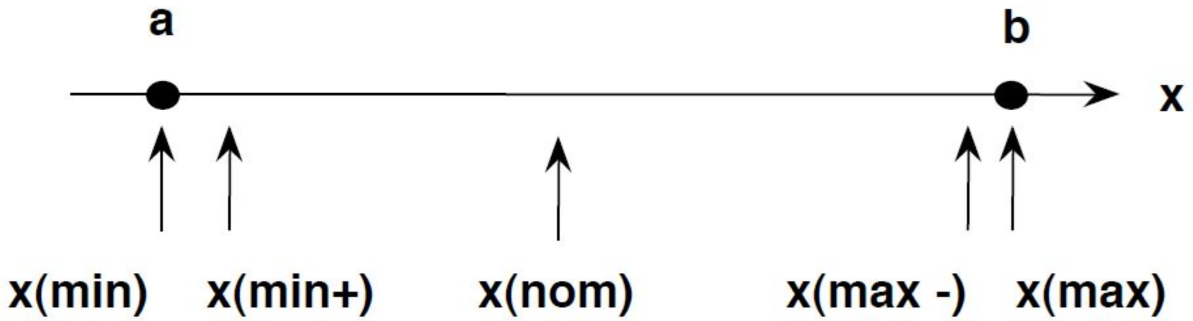
- “Coding unit tests take too much time”
- “I’m too busy fixing bugs to write tests”
- “Testing is boring, I need room to create man”
- “My code is **F l A w L e S s**”
- “I thought the QA guys did all the testing”
- “We’ll test after the code works”

How do we unit test?

- Some questions to ask yourself
    - How do we know what we should test?
    - How many tests should I write?
    - How do I know when I am finished testing?
- Answer - it varies, but there are some methods we have that can help us
    - Boundary value analysis
    - Equivalence class partitioning

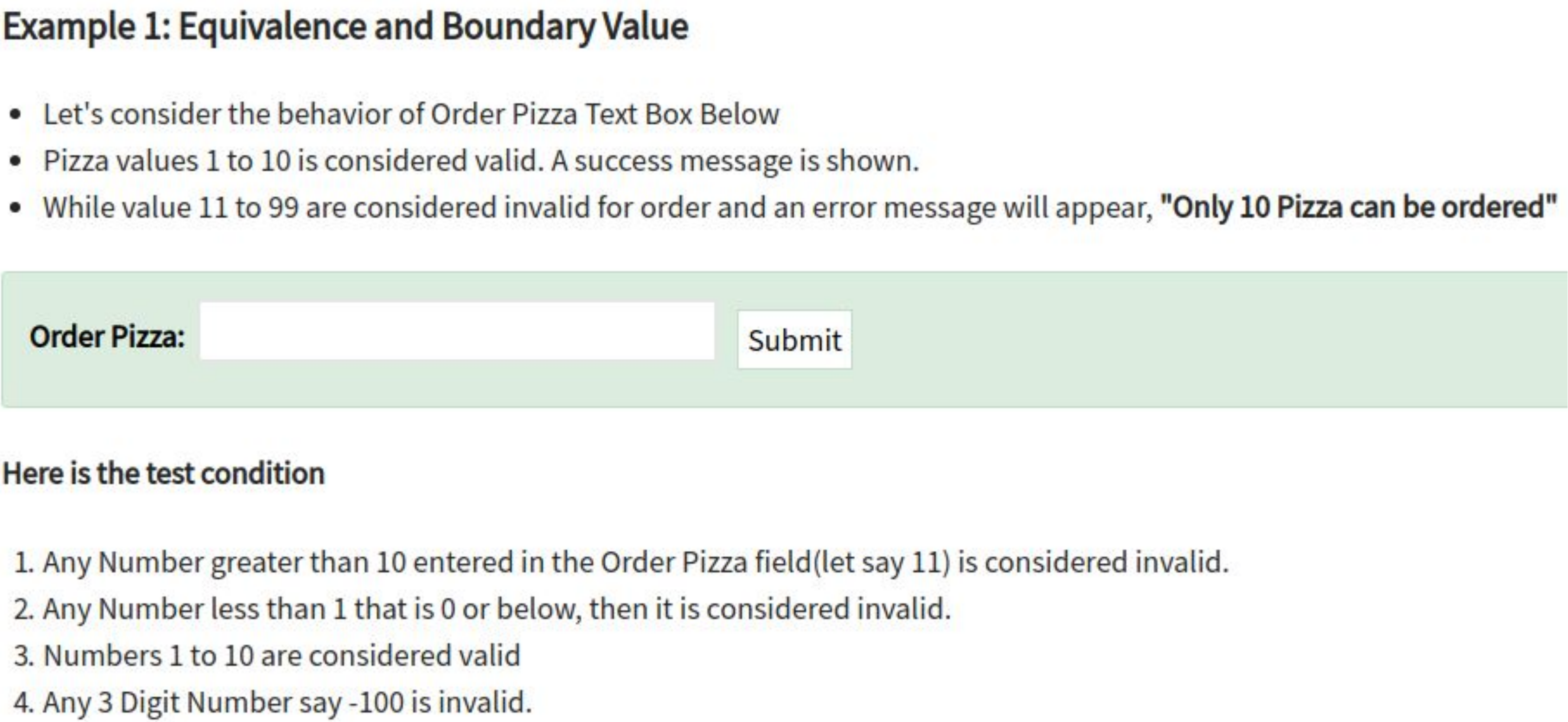
Boundary Testing

- Testing between extreme ends (i.e., boundaries)
- Select input variable values at their: Minimum, just above minimum, a typical value, just below maximum, maximum
- Before we do boundary testing, we need equivalence class partitioning



Equivalence Class Partitioning

- Divides input of software into equivalence data classes
    - Inputs fall into the same equivalence class IF they are part of the same same “type” of input. We will see some examples soon.
    - There can be multiple partitions, some of these will be partitions of valid types of data and others will be partitions of invalid types of data



Example 1: Equivalence and Boundary Value

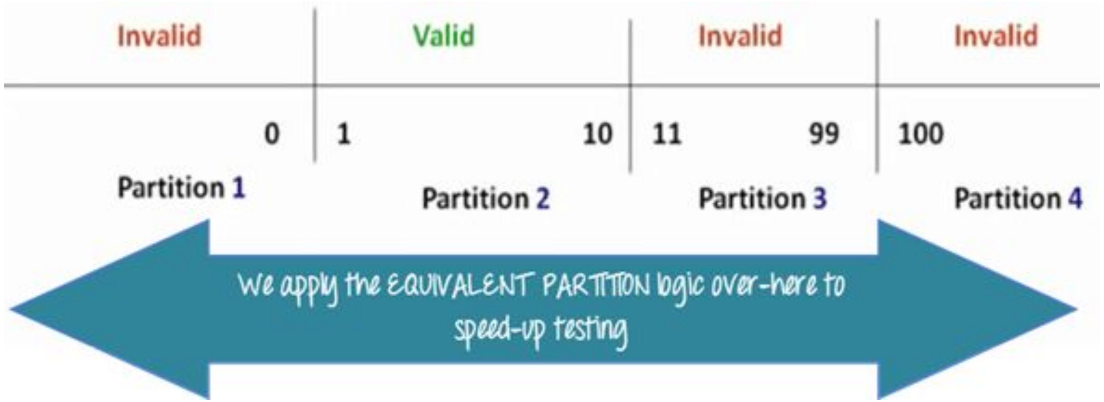
- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

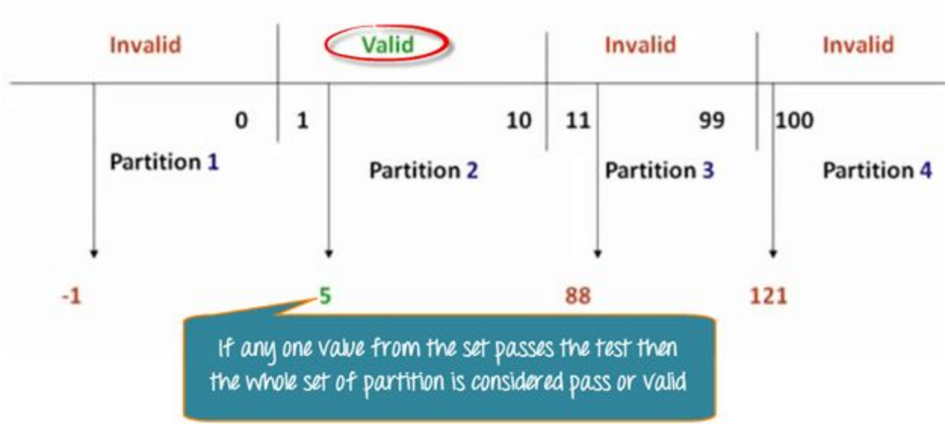
Why not test all values?

- Testing all values would be a waste of time and bloat the test set without any real value added
- But what do we do? We want to test around the points where things *miiight* get a little weird. So we use equivalence partitioning.



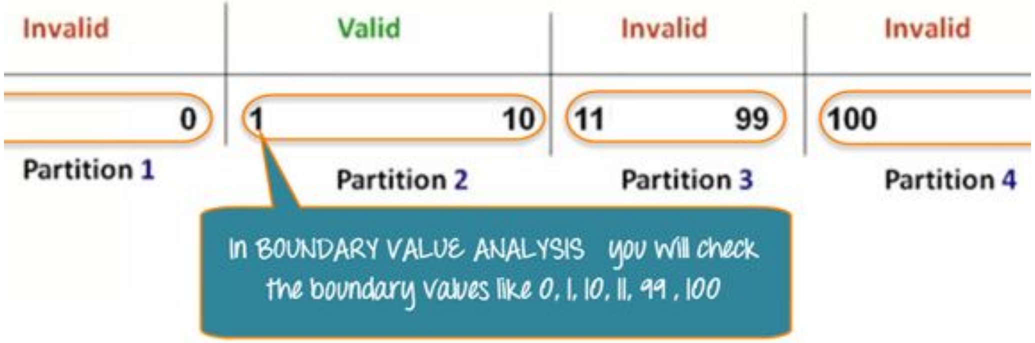
Apply equivalence partitioning

- The divided sets are equivalence partitions (or classes). Pick one value from each partition for testing.
  - The idea is that if one condition/value in a partition passes, the others will pass, too. Vice versa for failure.


Invalid
Valid
Invalid
Invalid
0
1
10
11
99
100
Partition 1
Partition 2
Partition 3
Partition 4
-1
5
88
121
If any one value from the set passes the test then the whole set of partition is considered pass or valid

Which value do we pick?

- We now apply boundary value analysis to test boundaries between equivalence partitions.



In BOUNDARY VALUE ANALYSIS you will check the boundary values like 0, 1, 10, 11, 99, 100

String Example

- We can do this with non-numeric data types, too. For example, assume our application reads a file and inserts into a linked list. Here are some tests we might write:
  - Node value cannot be empty string (outside of boundary)
  - Node value cannot contain uppercase characters (outside of boundary)
  - Node value cannot contain non-ascii characters (outside of boundary)
  - Node value cannot contain ONLY numeric characters (outside of boundary)
  - Node should accept all lowercase ascii characters (average case)

DB Example

- How do we do some of this for databases? Let’s say we are reading data into a database. What kind of partitions do we have?
    - Inserting valid data should create a record (average case)
    - Inserting invalid data should not create a record (outside of boundary)
    - If user is not authenticated, they should not get access to the database (outside boundary)
    - Reading (i.e., SELECT) from a database should return the appropriate records (average case)
    - Updating data in the DB should update appropriate records (average case)

Our partitions for Strings

Average case example: abc, class, rochester

Outside of Boundary cases: ‘’, 123, 😄, 😃, 🙃

Inside of boundary “edge”-case: 123abc

Conclusion

- Equivalence classes and Boundary Analysis are used to reduce large number of test cases to manageable chunks
- Provide clear guidelines for choosing test cases without compromising effectiveness

Images in presentation from:
https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html