



PEX

TABLE OF CONTENTS

01 What is parameterized unit testing?

02 Why use parameterized unit testing?

03 Overview of Pex

04 How to write parameterized unit tests
with pex

05 Example of Pex using .NET

WHAT IS PARAMETERIZED UNIT TESTING

- Parameterized unit testing is a testing technique that allows for testing a function or method with a variety of inputs and expected outputs
- The inputs can be any combination of values, types, or structures
- The expected outputs are the results that the function or method is expected to return for each input

WHY USE PARAMETERIZED UNIT TESTING

- To identify bugs and edge cases that may not be caught by testing a function with a single input
- By testing a function with a range of inputs, including edge cases, we can ensure that the function behaves correctly for all possible inputs
- The expected outputs are the results that the function or method is expected to return for each input
- Also save time and effort by automating the testing process .Instead of manually testing a function with a range of inputs, we can write a single parameterized test that tests the function with all possible inputs
- This can be particularly useful when testing functions with a large number of possible inputs


OVERVIEW OF PEX

- Pex is a popular tool for parameterized unit testing in .NET applications. Pex generates test cases automatically based on the function or method signature and the input constraints specified by the developer.
- Pex can also generate code coverage reports and identify code paths that are not covered by the tests.
- Pex uses a technique called symbolic execution to generate test cases. Symbolic execution is a method for analyzing a program by executing it with symbolic values instead of concrete values. By using symbolic values, Pex can explore all possible code paths and generate test cases that cover all possible inputs and edge cases.

HOW TO WRITE PARAMETERIZED UNIT TESTS WITH PEX

1. **Define the function or method to be tested:**
2. **Add input constraints:** We need to add input constraints to the function or method. Input constraints specify the range of values or types that the inputs can take. We can add input constraints using Pex attributes or by writing custom code.
3. **Generate tests:** We can generate tests using Pex by running the Pex explorer or by using the Pex API in our code.
4. **Review and refine the tests:** After generating the tests, we need to review and refine them. We can remove any redundant or unnecessary tests and add additional tests to cover edge cases.
5. **Run the tests:** We can run the tests using Visual Studio's Test Explorer or by using the Pex API in our code. The tests will run automatically, and the results will be displayed in the test explorer.
6. **Analyze the results:** After running the tests, we need to analyze the results to ensure that the function or method behaves correctly for all possible inputs. We can use the code coverage reports generated by Pex to identify code paths that are not covered by the tests.

EXAMPLE OF PARAMETERIZED UNIT TESTING WITH PEX



```
public static int Add(int a, int b)
{
    return a + b;
}
```

EXAMPLE OF PARAMETERIZED UNIT TESTING WITH PEX

```
public void TestAdd(int a, int b)
{
    PexAssume.IsTrue(a >= int.MinValue && a <= int.MaxValue);
    PexAssume.IsTrue(b >= int.MinValue && b <= int.MaxValue);

    int result = Add(a, b);

    PexAssert.AreEqual(result, a + b);
}
```


CONCLUSION

- Parameterized unit testing is a powerful technique for testing functions and methods with a range of inputs and expected outputs.
- By using Pex, we can automate the testing process and ensure that our functions and methods behave correctly for all possible inputs and edge cases.

THANK YOU