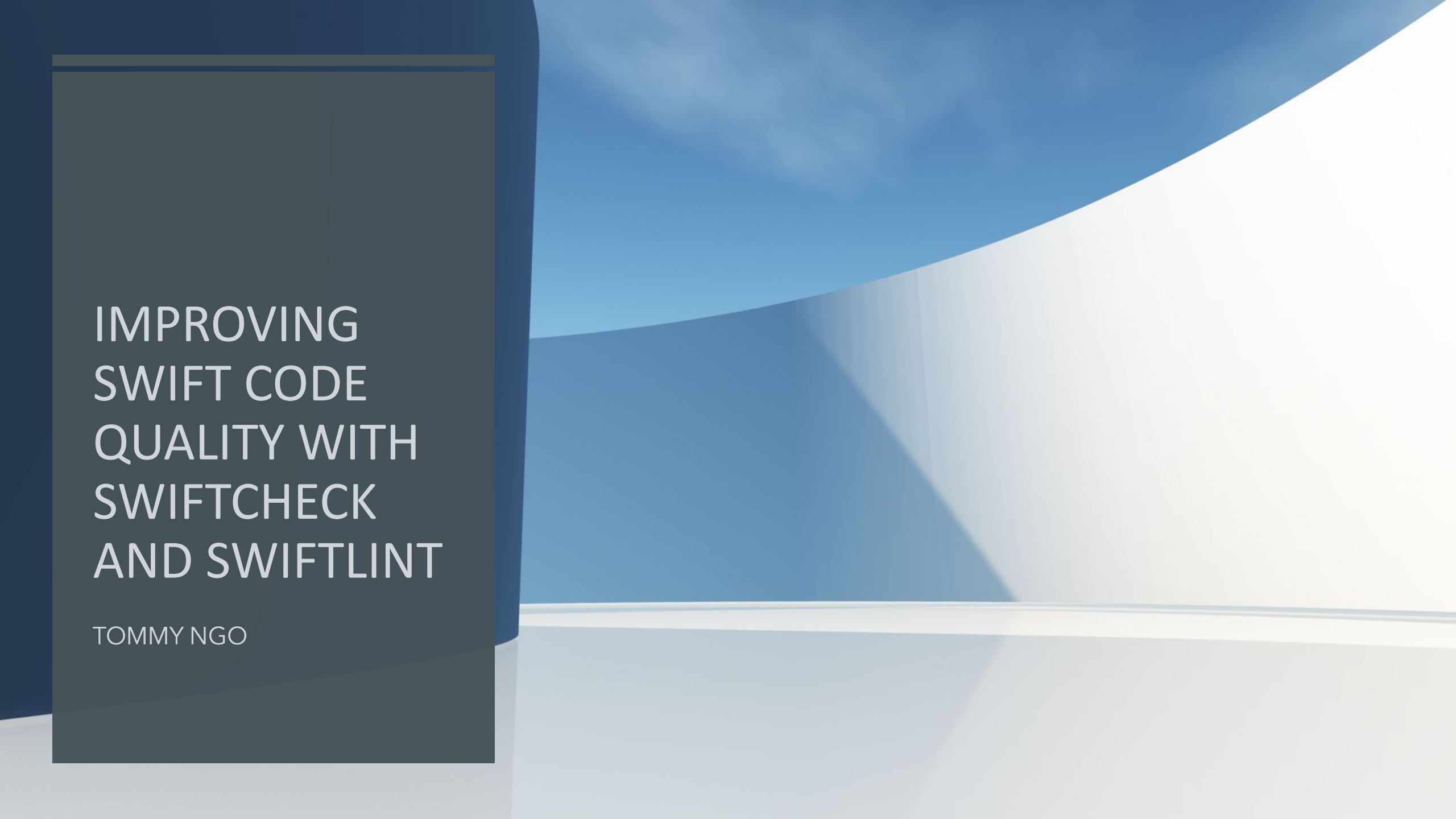

IMPROVING SWIFT CODE QUALITY WITH SWIFTCHECK AND SWIFTLINT

TOMMY NGO

INTRODUCTION

- Importance of writing maintainable, readable, and error-free code
- Overview of SwiftCheck and SwiftLint tools
- Goal of the presentation

SWIFTCHECK INTRODUCTION

- Brief explanation of what SwiftCheck is
- Explanation of property-based testing
- Benefits of property-based testing with SwiftCheck
- Example of a property-based test using SwiftCheck

SWIFTCHECK

- Example code for the addNumbers function:

```
func addNumbers(_ x: Int, _ y: Int) -> Int {
  return x + y
}
```

SWIFTCHECK ADVANTAGES

- Finding edge cases
- Comprehensive testing
- Example of testing a function with SwiftCheck

EXAMPLE OF TESTING A FUNCTION WITH SWIFTCHECK

- Example code for the addNumbers function:

```
func addNumbers(_ x: Int, _ y: Int) -> Int {
  return x + y
}
```

- Example code for the property-based test with SwiftCheck:

```
import XCTest
import SwiftCheck

class MyTests: XCTestCase {
  func testAddition() {
    property("The sum of two integers should always be greater than either of the operands")
    <- forAll { (x: Int, y: Int) in
      let sum = addNumbers(x, y)
      return sum > x && sum > y
    }
  }
}
```

- Example code for the calculateAverage function:

```
func calculateAverage(_ numbers: [Int]) -> Double {
  guard !numbers.isEmpty else { return 0 }
  let sum = numbers.reduce(0, +)
  return Double(sum) / Double(numbers.count)
}
```

- Example code for the property-based test with SwiftCheck:

```
func testCalculateAverage() {
  property("The average of an array of integers should be the
                sum of integers divided by the count of integers")
      <- forAll { (numbers: [Int]) in
    guard !numbers.isEmpty else { return true }
    let sum = numbers.reduce(0, +)
    let expectedAverage = Double(sum) / Double(numbers.count)
    let actualAverage = calculateAverage(numbers)
    return abs(expectedAverage - actualAverage) < 0.001
  }
}
```

SWIFTCHECK DISADVANTAGES

- Time-consuming testing for large or complex functions
- Tips to mitigate time consumption

SWIFTLINT INTRODUCTION

- Brief explanation of what SwiftLint is
- Importance of enforcing coding standards
- How SwiftLint enforces coding standards
- Example of a rule in SwiftLint

SWIFTLINT ADVANTAGES

- Consistent coding style
- Improved readability
- Example of a rule for improving readability

EXAMPLE OF A RULE IN SWIFTLINT

- Example code for the **.swiftlint.yml** file to enforce the naming convention:

```
function_name:
  pattern: "^[a-z]+([A-Z][a-z]+)*$"
```

- Example code for the if statements with a violation of the coding standard:

```
if condition1 {
  if condition2 {
    // Do something
  }
}
```

EXAMPLE OF A RULE IN SWIFTLINT

- Example code for the .swiftlint.yml file to enforce the coding standard:

```
function_name:
  pattern: "^[a-z]+([A-Z][a-z]+)*$"
```

- Example code for the if statements with a violation of the coding standard:

```
if_else_collapse:
  severity: warning
  ignored_contexts: [ "SwitchStatement" ]
  inverted_condition: true
```

PRACTICAL EXAMPLES WITH XCODE

- Demonstrate how to install SwiftCheck and SwiftLint in XCode projects
- Walk through creating property-based tests with SwiftCheck
- Show how to set up and configure SwiftLint in XCode projects
- Provide examples of customizing rules in SwiftLint

CONCLUSION

- Recap of SwiftCheck and SwiftLint features and benefits
- Importance of maintaining high code quality
- Encourage use of SwiftCheck and SwiftLint to improve code quality

Q&A

- Encourage attendees to ask questions and provide feedback
- Provide contact information for the presenter if necessary