Developer Tools

#WWDC17

Engineering for Testability

Session 414

Brian Croom, Xcode Engineer
Greg Tracy, Xcode Engineer

Engineering for Testability

Testable app code

Scalable test code

Engineering for Testability

Testable app code

Scalable test code

Engineering for Testability

Testable app code

Scalable test code

Testable App Code

Structure of a Unit Test

Structure of a Unit Test

```
func testArraySorting() {
  let input = [1, 7, 6, 3, 10]

  let output = input.sorted()

  XCTAssertEqual(output, [1, 3, 6, 7, 10])
}
```

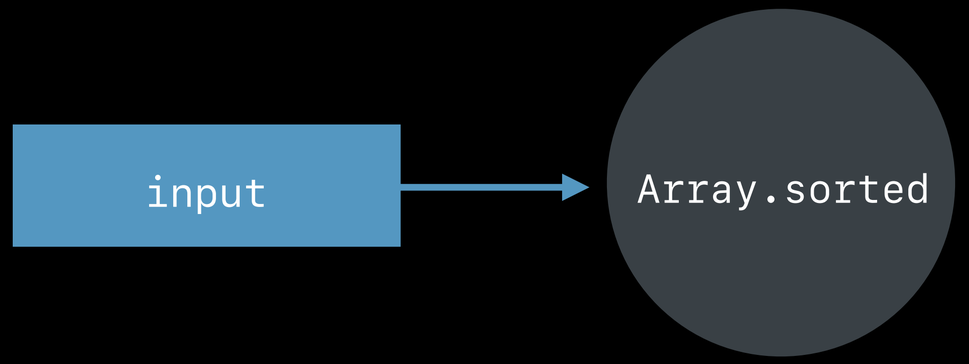
Structure of a Unit Test

```
func testArraySorting() {
  let input = [1, 7, 6, 3, 10]

  let output = input.sorted()

  XCTAssertEqual(output, [1, 3, 6, 7, 10])
}
```

input

Structure of a Unit Test

```
func testArraySorting() {
  let input = [1, 7, 6, 3, 10]

  let output = input.sorted()

  XCTAssertEqual(output, [1, 3, 6, 7, 10])
}
```



Structure of a Unit Test

```
func testArraySorting() {
  let input = [1, 7, 6, 3, 10]

  let output = input.sorted()

  XCTAssertEqual(output, [1, 3, 6, 7, 10])
}
```

Structure of a Unit Test

Structure of a Unit Test

Prepare input

Structure of a Unit Test

Prepare input

Run the code being tested

Structure of a Unit Test

Prepare input

Run the code being tested

Verify output

Characteristics of Testable Code

Characteristics of Testable Code

Control over inputs

Characteristics of Testable Code

Control over inputs

Visibility into outputs

Characteristics of Testable Code

Control over inputs

Visibility into outputs

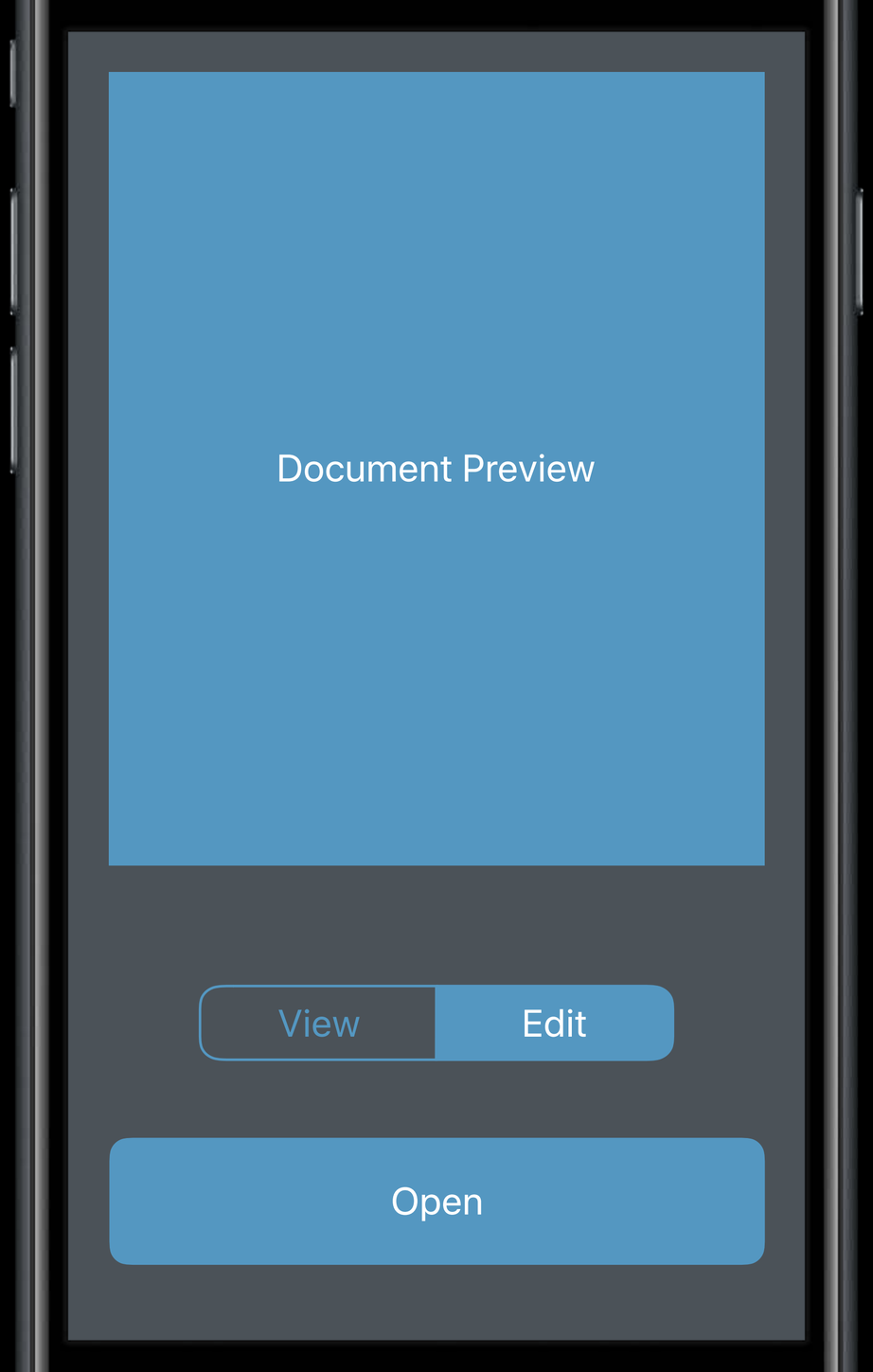
No hidden state

Testability Techniques

Protocols and parameterization

Separating logic and effects

Protocols and Parameterization


Document Preview
View
Edit
Open

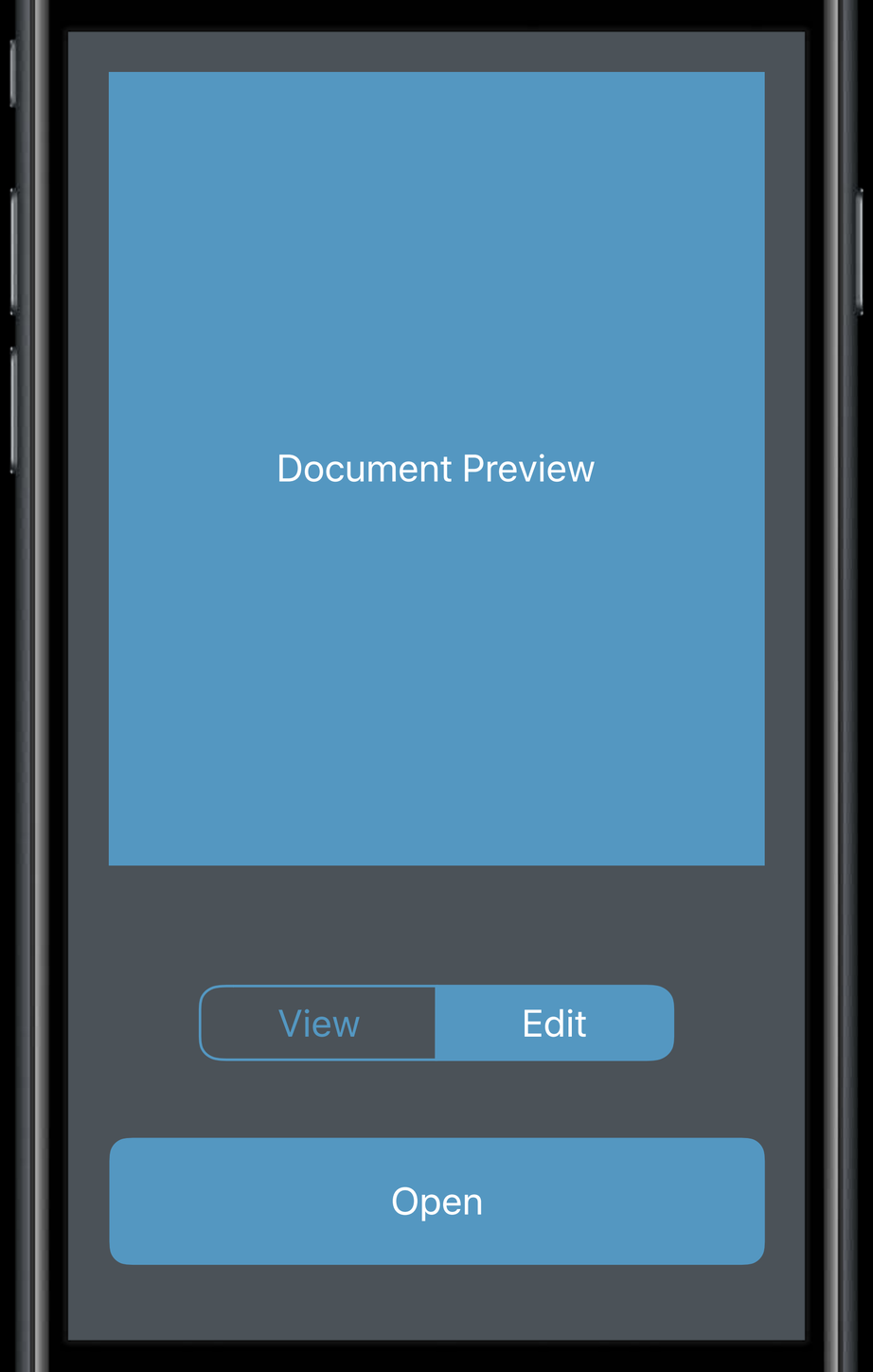
```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```


Document Preview
View
Edit
Open

Document Editor App

```swift
func testOpensDocumentURLWhenButtonIsTapped() {



}
```

```
func testOpensDocumentURLWhenButtonIsTapped() {
  let controller = UIStoryboard(name: "Main", bundle: nil)
      .instantiateViewController(withIdentifier: "Preview") as! PreviewViewController




}
```

```
func testOpensDocumentURLWhenButtonIsTapped() {
  let controller = UIStoryboard(name: "Main", bundle: nil)
      .instantiateViewController(withIdentifier: "Preview") as! PreviewViewController
  controller.loadViewIfNeeded()





}
```

```
func testOpensDocumentURLWhenButtonIsTapped() {
  let controller = UIStoryboard(name: "Main", bundle: nil)
      .instantiateViewController(withIdentifier: "Preview") as! PreviewViewController
  controller.loadViewIfNeeded()
  controller.segmentedControl.selectedSegmentIndex = 1





}
```

```
func testOpensDocumentURLWhenButtonIsTapped() {
  let controller = UIStoryboard(name: "Main", bundle: nil)
      .instantiateViewController(withIdentifier: "Preview") as! PreviewViewController
  controller.loadViewIfNeeded()
  controller.segmentedControl.selectedSegmentIndex = 1
  controller.document = Document(identifier: "TheID")




}
```

```
func testOpensDocumentURLWhenButtonIsTapped() {
  let controller = UIStoryboard(name: "Main", bundle: nil)
      .instantiateViewController(withIdentifier: "Preview") as! PreviewViewController
  controller.loadViewIfNeeded()
  controller.segmentedControl.selectedSegmentIndex = 1
  controller.document = Document(identifier: “TheID")

  controller.openTapped(controller.button)


}
```

```
func testOpensDocumentURLWhenButtonIsTapped() {
  let controller = UIStoryboard(name: "Main", bundle: nil)
      .instantiateViewController(withIdentifier: "Preview") as! PreviewViewController
  controller.loadViewIfNeeded()
  controller.segmentedControl.selectedSegmentIndex = 1
  controller.document = Document(identifier: “TheID")

  controller.openTapped(controller.button)

  // ???
}
```

```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```swift
@IBAction func openTapped(_ sender: Any) {
  let mode: String
  switch segmentedControl.selectedSegmentIndex {
  case 0: mode = "view"
  case 1: mode = "edit"
  default: fatalError("Impossible case")
  }
  let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(mode)")!

  if UIApplication.shared.canOpenURL(url) {
    UIApplication.shared.open(url, options: [:], completionHandler: nil)
  } else {
    handleURLError()
  }
}
```

```swift
class DocumentOpener {
  enum OpenMode: String {
    case view
    case edit
  }
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if UIApplication.shared.canOpenURL(url) {
      UIApplication.shared.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class DocumentOpener {
  enum OpenMode: String {
    case view
    case edit
  }
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if UIApplication.shared.canOpenURL(url) {
      UIApplication.shared.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class DocumentOpener {
  enum OpenMode: String {
    case view
    case edit
  }
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if UIApplication.shared.canOpenURL(url) {
      UIApplication.shared.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class DocumentOpener {
  enum OpenMode: String {
    case view
    case edit
  }
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if UIApplication.shared.canOpenURL(url) {
      UIApplication.shared.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class DocumentOpener {
  enum OpenMode: String {
    case view
    case edit
  }
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if UIApplication.shared.canOpenURL(url) {
      UIApplication.shared.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class DocumentOpener {
  let application: UIApplication
  init(application: UIApplication) {
    self.application = application
  }

  /* … */
}
```

```swift
class DocumentOpener {
  let application: UIApplication
  init(application: UIApplication = UIApplication.shared) {
    self.application = application
  }

  /* … */
}
```

```swift
class DocumentOpener {
  /* … */
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if application.canOpenURL(url) {
      application.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class DocumentOpener {
  /* … */
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if application.canOpenURL(url) {
      application.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
func testDocumentOpenerWhenItCanOpen() {
  let app = /* ??? */
  let opener = DocumentOpener(application: app)




}
```

```swift
protocol URLOpening {


}
```

```swift
protocol URLOpening {
  func canOpenURL(_ url: URL) -> Bool
  func open(_ url: URL, options: [String: Any], completionHandler: ((Bool) -> Void)?)
}
```

```swift
protocol URLOpening {
  func canOpenURL(_ url: URL) -> Bool
  func open(_ url: URL, options: [String: Any], completionHandler: ((Bool) -> Void)?)
}

extension UIApplication: URLOpening {

}
```

```swift
protocol URLOpening {
  func canOpenURL(_ url: URL) -> Bool
  func open(_ url: URL, options: [String: Any], completionHandler: ((Bool) -> Void)?)
}

extension UIApplication: URLOpening {
  // Nothing needed here!
}
```

```swift
class DocumentOpener {
  let application: UIApplication
  init(application: UIApplication = UIApplication.shared) {
    self.application = application
  }

  /* … */
}
```

```swift
class DocumentOpener {
  let urlOpener: URLOpening
  init(urlOpener: URLOpening = UIApplication.shared) {
    self.urlOpener = urlOpener
  }

  /* … */
}
```

```swift
class DocumentOpener {
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if application.canOpenURL(url) {
      application.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class DocumentOpener {
  func open(_ document: Document, mode: OpenMode) {
    let modeString = mode.rawValue
    let url = URL(string: "myappscheme://open?id=\(document.identifier)&mode=\(modeString)")!

    if urlOpener.canOpenURL(url) {
      urlOpener.open(url, options: [:], completionHandler: nil)
    } else {
      handleURLError()
    }
  }
}
```

```swift
class MockURLOpener: URLOpening {

}
```

```
class MockURLOpener: URLOpening {

  func canOpenURL(_ url: URL) -> Bool {

  }

  func open(_ url: URL,
            options: [String: Any],
            completionHandler: ((Bool) -> Void)?) {

  }
}
```

```swift
class MockURLOpener: URLOpening {
  var canOpen = false


  func canOpenURL(_ url: URL) -> Bool {
    return canOpen
  }

  func open(_ url: URL,
            options: [String: Any],
            completionHandler: ((Bool) -> Void)?) {

  }
}
```

```swift
class MockURLOpener: URLOpening {
  var canOpen = false
  var openedURL: URL?

  func canOpenURL(_ url: URL) -> Bool {
    return canOpen
  }

  func open(_ url: URL,
            options: [String: Any],
            completionHandler: ((Bool) -> Void)?) {
    openedURL = url
  }
}
```

```
func testDocumentOpenerWhenItCanOpen() {



}
```

```
func testDocumentOpenerWhenItCanOpen() {
  let urlOpener = MockURLOpener()
  urlOpener.canOpen = true




}
```

```swift
func testDocumentOpenerWhenItCanOpen() {
  let urlOpener = MockURLOpener()
  urlOpener.canOpen = true
  let documentOpener = DocumentOpener(urlOpener: urlOpener)



}
```

```swift
func testDocumentOpenerWhenItCanOpen() {
  let urlOpener = MockURLOpener()
  urlOpener.canOpen = true
  let documentOpener = DocumentOpener(urlOpener: urlOpener)

  documentOpener.open(Document(identifier: "TheID"), mode: .edit)


}
```

```swift
func testDocumentOpenerWhenItCanOpen() {
  let urlOpener = MockURLOpener()
  urlOpener.canOpen = true
  let documentOpener = DocumentOpener(urlOpener: urlOpener)

  documentOpener.open(Document(identifier: "TheID"), mode: .edit)

  XCTAssertEqual(urlOpener.openedURL, URL(string: "myappscheme://open?id=TheID&mode=edit"))
}
```

```swift
func testDocumentOpenerWhenItCanOpen() {
  let urlOpener = MockURLOpener()
  urlOpener.canOpen = true
  let documentOpener = DocumentOpener(urlOpener: urlOpener)

  documentOpener.open(Document(identifier: "TheID"), mode: .edit)

  XCTAssertEqual(urlOpener.openedURL, URL(string: "myappscheme://open?id=TheID&mode=edit"))
}
```

Protocols and Parameterization

Protocols and Parameterization

Reduce references to shared instances

Protocols and Parameterization

Reduce references to shared instances

Accept parameterized input

Protocols and Parameterization

Reduce references to shared instances

Accept parameterized input

Introduce a protocol

Protocols and Parameterization

Reduce references to shared instances

Accept parameterized input

Introduce a protocol

Create a testing implementation

Separating Logic and Effects

```swift
class OnDiskCache {

}
```

```swift
class OnDiskCache {
  struct Item {
    let path: String
    let age: TimeInterval
    let size: Int
  }

  var currentItems: Set<Item> { /* … */ }

  /* … */
}
```

```swift
class OnDiskCache {
  struct Item {
    let path: String
    let age: TimeInterval
    let size: Int
  }

  var currentItems: Set<Item> { /* … */ }

  /* … */
}
```

```swift
class OnDiskCache {
  /* … */

  func cleanCache(maxSize: Int) throws {









  }
}
```

```swift
class OnDiskCache {
  /* … */

  func cleanCache(maxSize: Int) throws {
    let sortedItems = self.currentItems.sorted { $0.age < $1.age }









  }
}
```

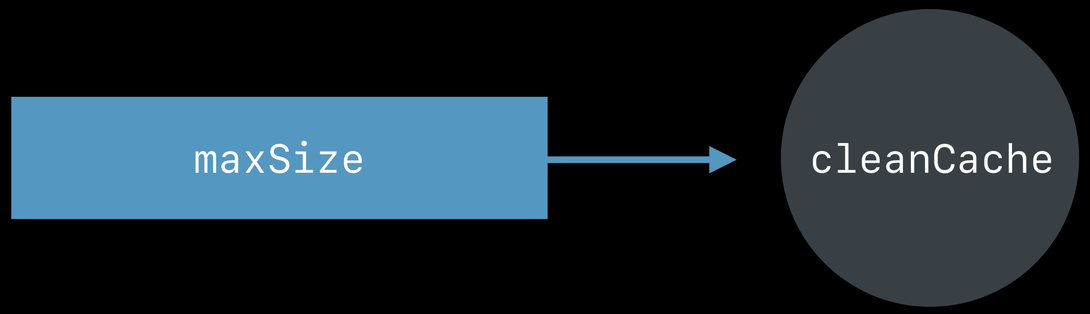
```swift
class OnDiskCache {
  /* … */

  func cleanCache(maxSize: Int) throws {
    let sortedItems = self.currentItems.sorted { $0.age < $1.age }

    var cumulativeSize = 0
    for item in sortedItems {





    }
  }
}
```

```swift
class OnDiskCache {
  /* … */

  func cleanCache(maxSize: Int) throws {
    let sortedItems = self.currentItems.sorted { $0.age < $1.age }

    var cumulativeSize = 0
    for item in sortedItems {
      cumulativeSize += item.size




    }
  }
}
```

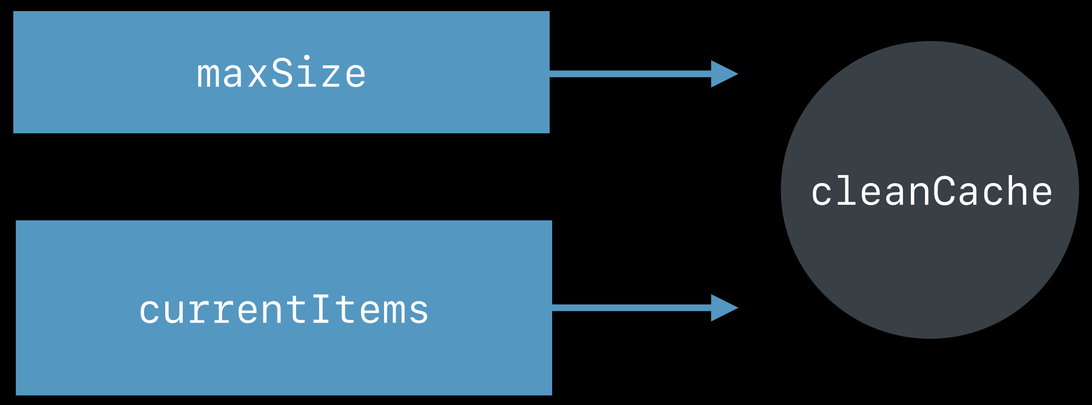
```swift
class OnDiskCache {
  /* … */

  func cleanCache(maxSize: Int) throws {
    let sortedItems = self.currentItems.sorted { $0.age < $1.age }

    var cumulativeSize = 0
    for item in sortedItems {
      cumulativeSize += item.size
      if cumulativeSize > maxSize {
        try FileManager.default.removeItem(atPath: item.path)
      }
    }
  }
}
```

```swift
class OnDiskCache {
  /* … */

  func cleanCache(maxSize: Int) throws {
    let sortedItems = self.currentItems.sorted { $0.age < $1.age }

    var cumulativeSize = 0
    for item in sortedItems {
      cumulativeSize += item.size
      if cumulativeSize > maxSize {
        try FileManager.default.removeItem(atPath: item.path)
      }
    }
  }
}
```

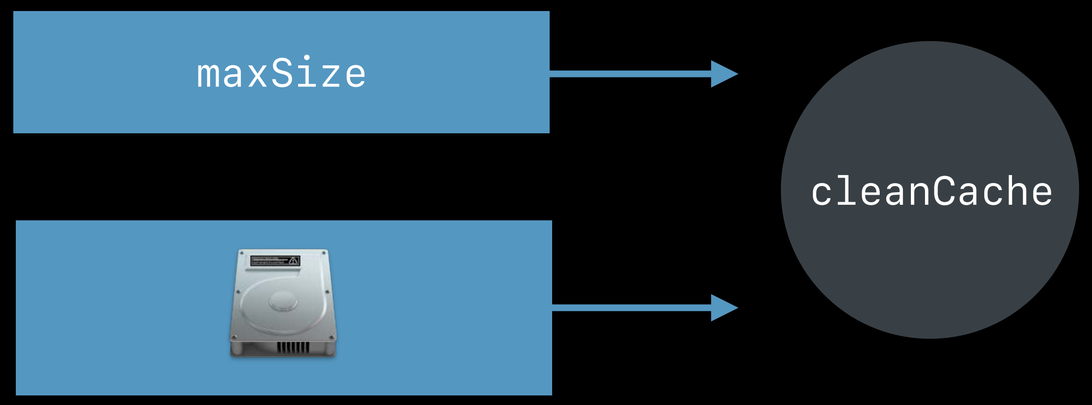
Inputs and Outputs



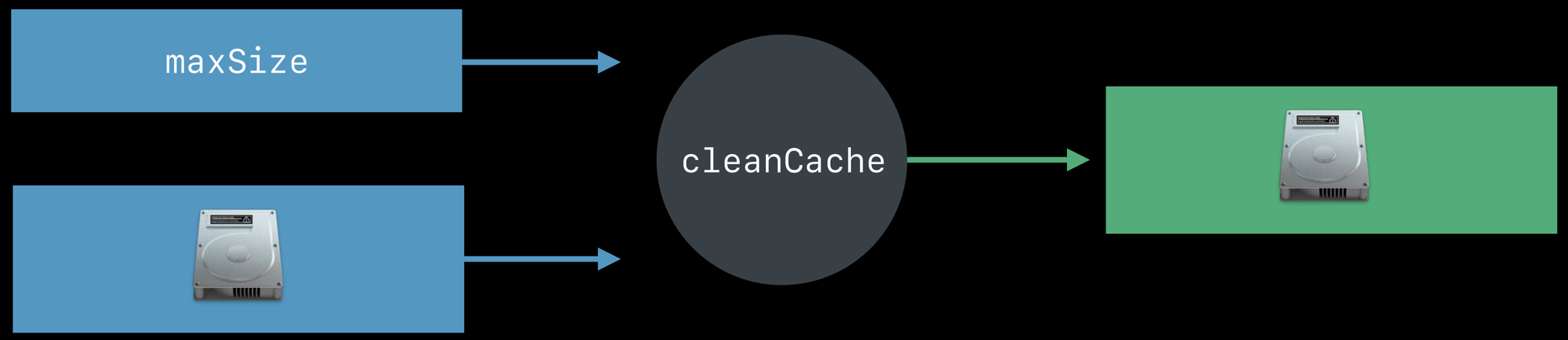
Inputs and Outputs



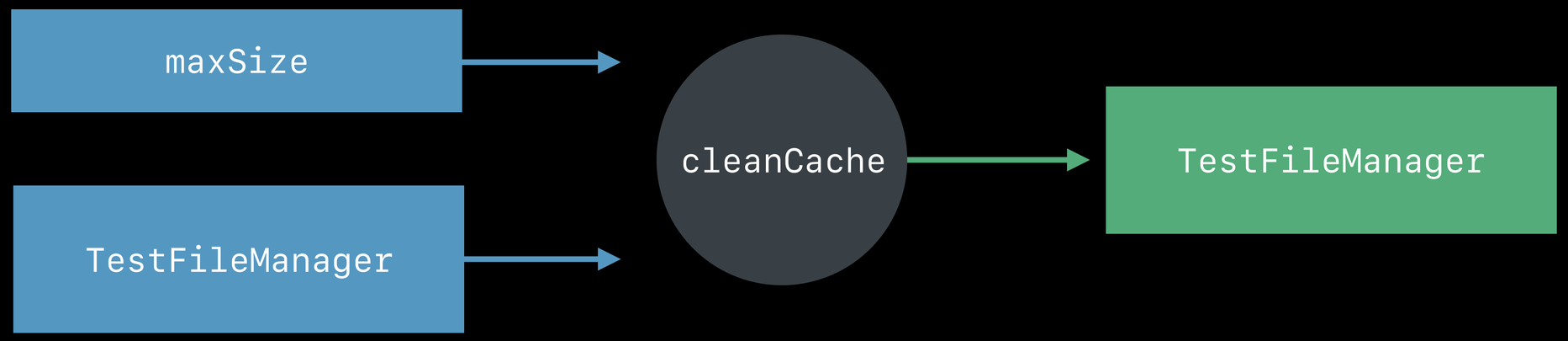
Inputs and Outputs



Inputs and Outputs

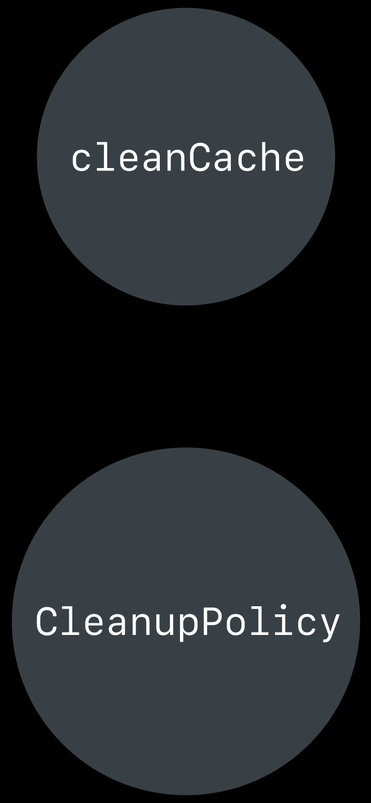


Inputs and Outputs



Inputs and Outputs

Inputs and Outputs



```swift
protocol CleanupPolicy {

}
```

```swift
protocol CleanupPolicy {
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item>
}
```

```swift
protocol CleanupPolicy {
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item>
}

class OnDiskCache {
  func cleanCache(maxSize: Int) throws { /* … */ }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {

}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int

}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {

  }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {
    var itemsToRemove = Set<OnDiskCache.Item>()
    var cumulativeSize = 0









    return itemsToRemove
  }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {
    var itemsToRemove = Set<OnDiskCache.Item>()
    var cumulativeSize = 0

    let sortedItems = items.sorted { $0.age < $1.age }
    for item in sortedItems {





    }
    return itemsToRemove
  }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {
    var itemsToRemove = Set<OnDiskCache.Item>()
    var cumulativeSize = 0

    let sortedItems = allItems.sorted { $0.age < $1.age }
    for item in sortedItems {
      cumulativeSize += item.size




    }
    return itemsToRemove
  }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {
    var itemsToRemove = Set<OnDiskCache.Item>()
    var cumulativeSize = 0

    let sortedItems = allItems.sorted { $0.age < $1.age }
    for item in sortedItems {
      cumulativeSize += item.size
      if cumulativeSize > maxSize {
        itemsToRemove.insert(item)
      }
    }
    return itemsToRemove
  }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {
    var itemsToRemove = Set<OnDiskCache.Item>()
    var cumulativeSize = 0

    let sortedItems = allItems.sorted { $0.age < $1.age }
    for item in sortedItems {
      cumulativeSize += item.size
      if cumulativeSize > maxSize {
        itemsToRemove.insert(item)
      }
    }
    return itemsToRemove
  }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {
    var itemsToRemove = Set<OnDiskCache.Item>()
    var cumulativeSize = 0

    let sortedItems = allItems.sorted { $0.age < $1.age }
    for item in sortedItems {
      cumulativeSize += item.size
      if cumulativeSize > maxSize {
        itemsToRemove.insert(item)
      }
    }
    return itemsToRemove
  }
}
```

```swift
struct MaxSizeCleanupPolicy: CleanupPolicy {
  let maxSize: Int
  func itemsToRemove(from items: Set<OnDiskCache.Item>) -> Set<OnDiskCache.Item> {
    var itemsToRemove = Set<OnDiskCache.Item>()
    var cumulativeSize = 0

    let sortedItems = allItems.sorted { $0.age < $1.age }
    for item in sortedItems {
      cumulativeSize += item.size
      if cumulativeSize > maxSize {
        itemsToRemove.insert(item)
      }
    }
    return itemsToRemove
  }
}
```

Inputs and Outputs

```go
func testMaxSizeCleanupPolicy() {

}
```

```swift
func testMaxSizeCleanupPolicy() {
  let inputItems = Set([
    OnDiskCache.Item(path: "/item1", age: 5, size: 7),
    OnDiskCache.Item(path: "/item2", age: 3, size: 2),
    OnDiskCache.Item(path: "/item3", age: 9, size: 9)
  ])





}
```

```swift
func testMaxSizeCleanupPolicy() {
  let inputItems = Set([
    OnDiskCache.Item(path: "/item1", age: 5, size: 7),
    OnDiskCache.Item(path: "/item2", age: 3, size: 2),
    OnDiskCache.Item(path: "/item3", age: 9, size: 9)
  ])

  let outputItems =
    MaxSizeCleanupPolicy(maxSize: 10).itemsToRemove(from: inputItems)


}
```

```swift
func testMaxSizeCleanupPolicy() {
  let inputItems = Set([
    OnDiskCache.Item(path: "/item1", age: 5, size: 7),
    OnDiskCache.Item(path: "/item2", age: 3, size: 2),
    OnDiskCache.Item(path: "/item3", age: 9, size: 9)
  ])

  let outputItems =
    MaxSizeCleanupPolicy(maxSize: 10).itemsToRemove(from: inputItems)

  XCTAssertEqual(outputItems, [OnDiskCache.Item(path: "/item3", age: 9, size: 9)])
}
```

```swift
class OnDiskCache {
  /* … */

  func cleanCache(policy: CleanupPolicy) throws {
    let itemsToRemove = policy.itemsToRemove(from: self.currentItems)
    for item in itemsToRemove {
      try FileManager.default.removeItem(atPath: item.path)
    }
  }
}
```

```swift
class OnDiskCache {
  /* … */

  func cleanCache(policy: CleanupPolicy) throws {
    let itemsToRemove = policy.itemsToRemove(from: self.currentItems)
    for item in itemsToRemove {
      try FileManager.default.removeItem(atPath: item.path)
    }
  }
}
```

```swift
class OnDiskCache {
  /* … */

  func cleanCache(policy: CleanupPolicy) throws {
    let itemsToRemove = policy.itemsToRemove(from: self.currentItems)
    for item in itemsToRemove {
      try FileManager.default.removeItem(atPath: item.path)
    }
  }
}
```

Separating Logic and Effects

Separating Logic and Effects

Extract algorithms

Separating Logic and Effects

Extract algorithms

Functional style with value types

Separating Logic and Effects

Extract algorithms

Functional style with value types

Thin layer on top to execute effects

Testability Techniques

Protocols and parameterization

Separating logic and effects

Scalable Test Code

Greg Tracy, Xcode Engineer

Balance between UI and unit tests

Balance between UI and unit tests

Code to help UI tests scale

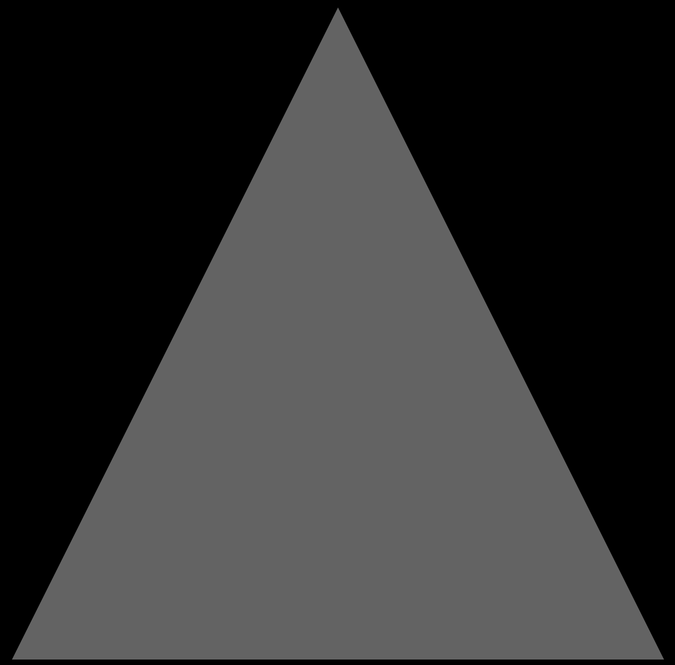
Balance between UI and unit tests

Code to help UI tests scale

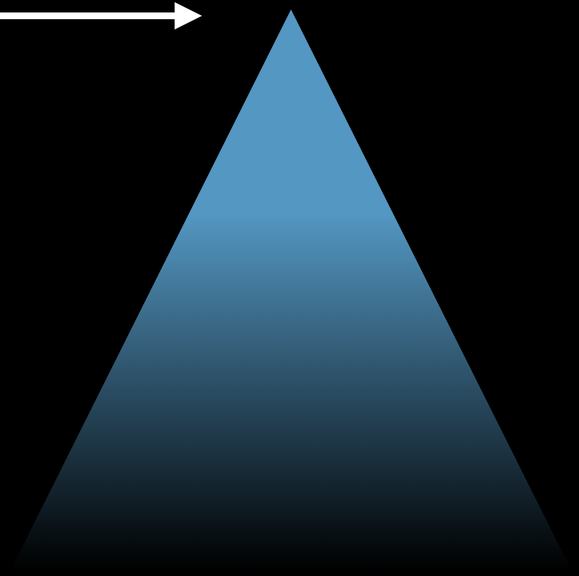
Quality of test code

Striking the right balance between UI and unit tests

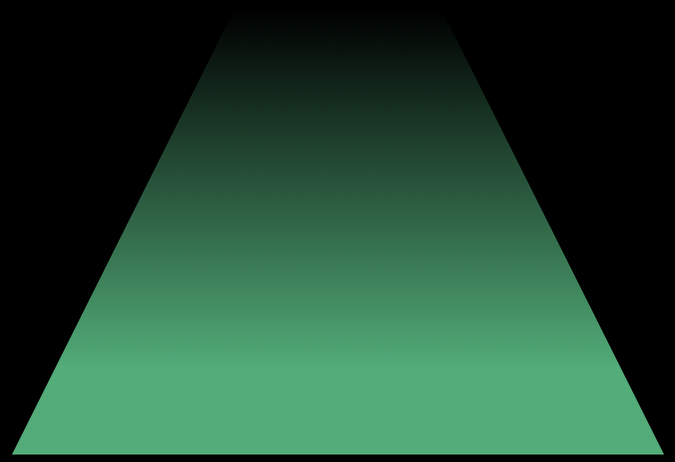
Testing Pyramid



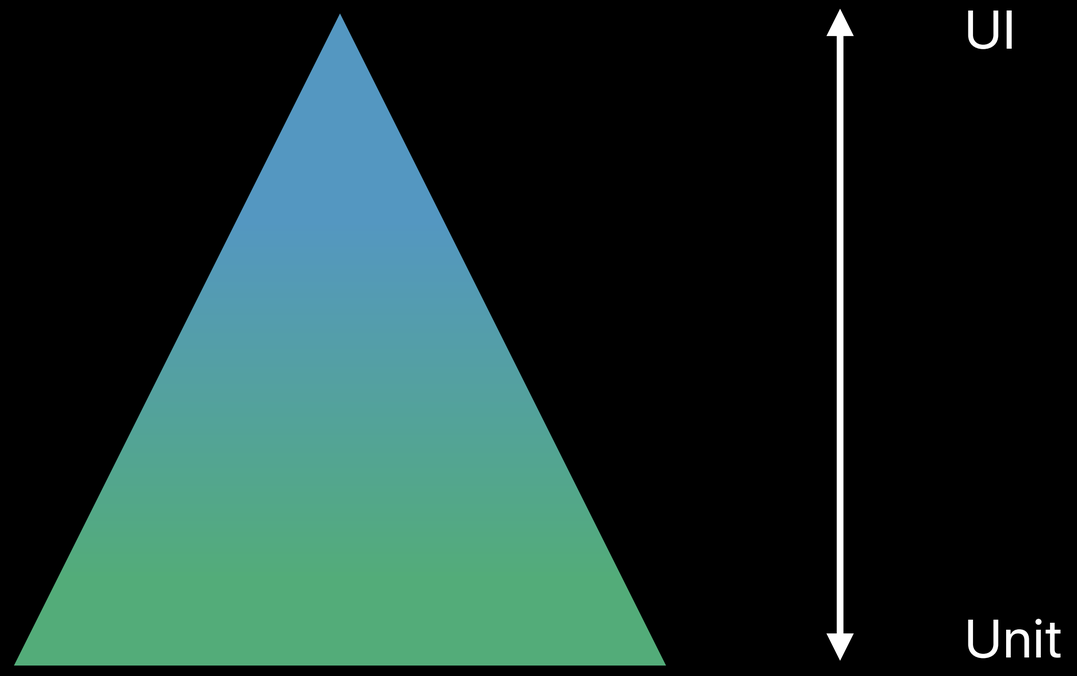
UI Tests



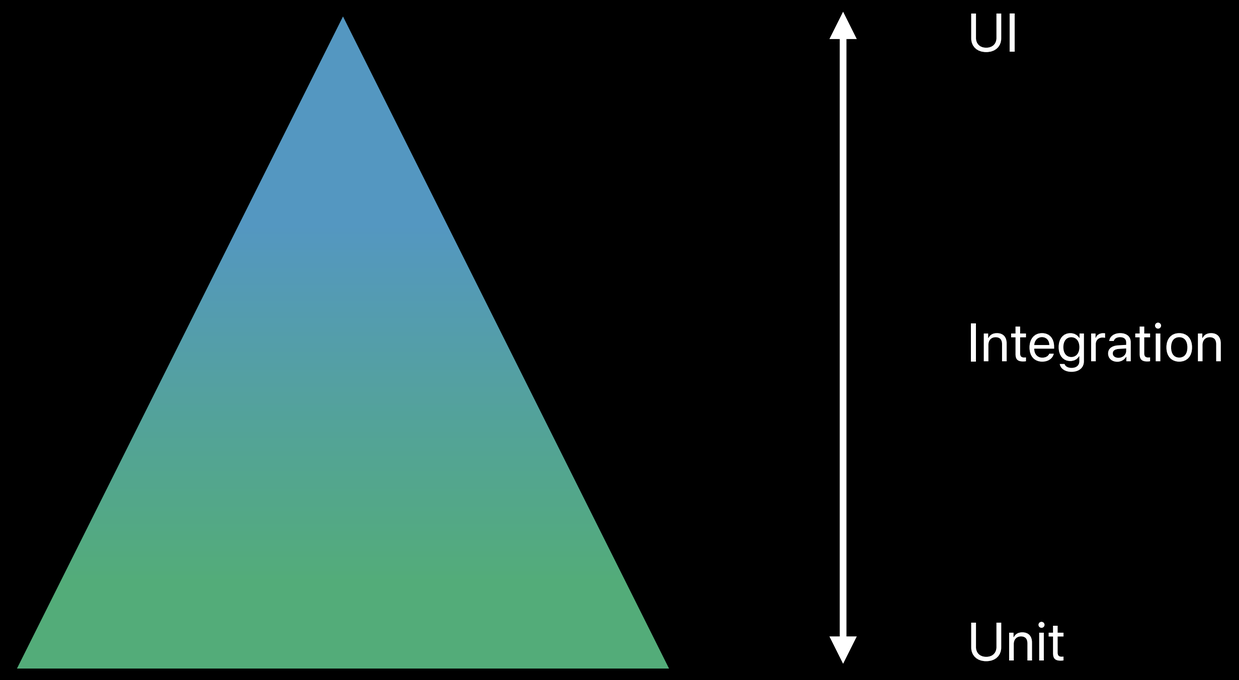
Unit Tests



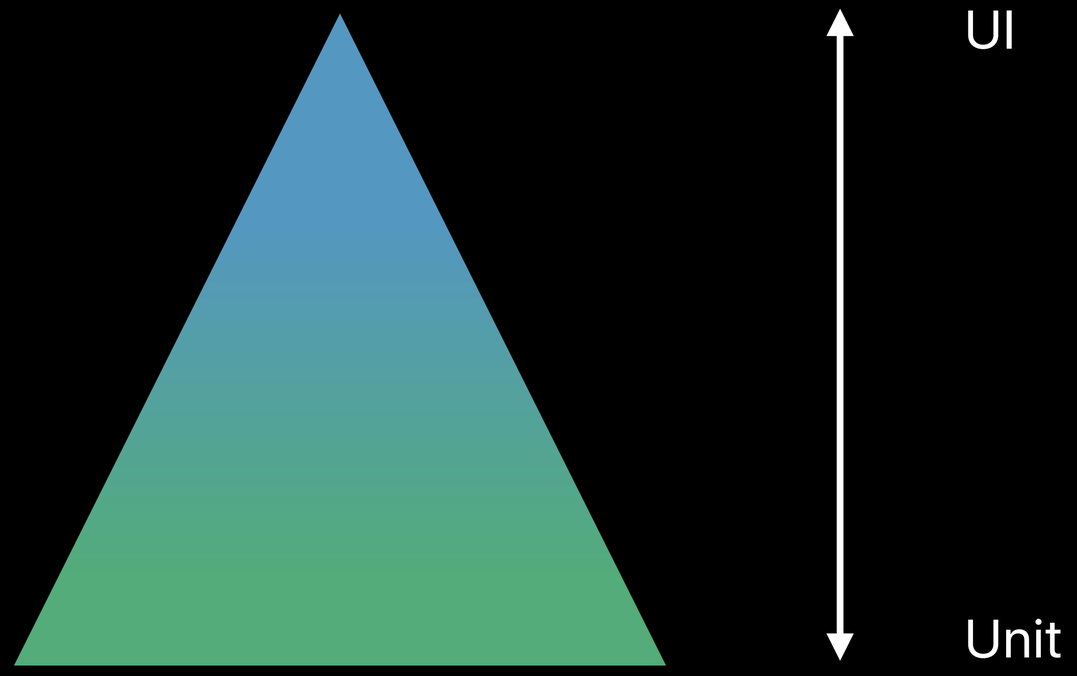
Testing Pyramid



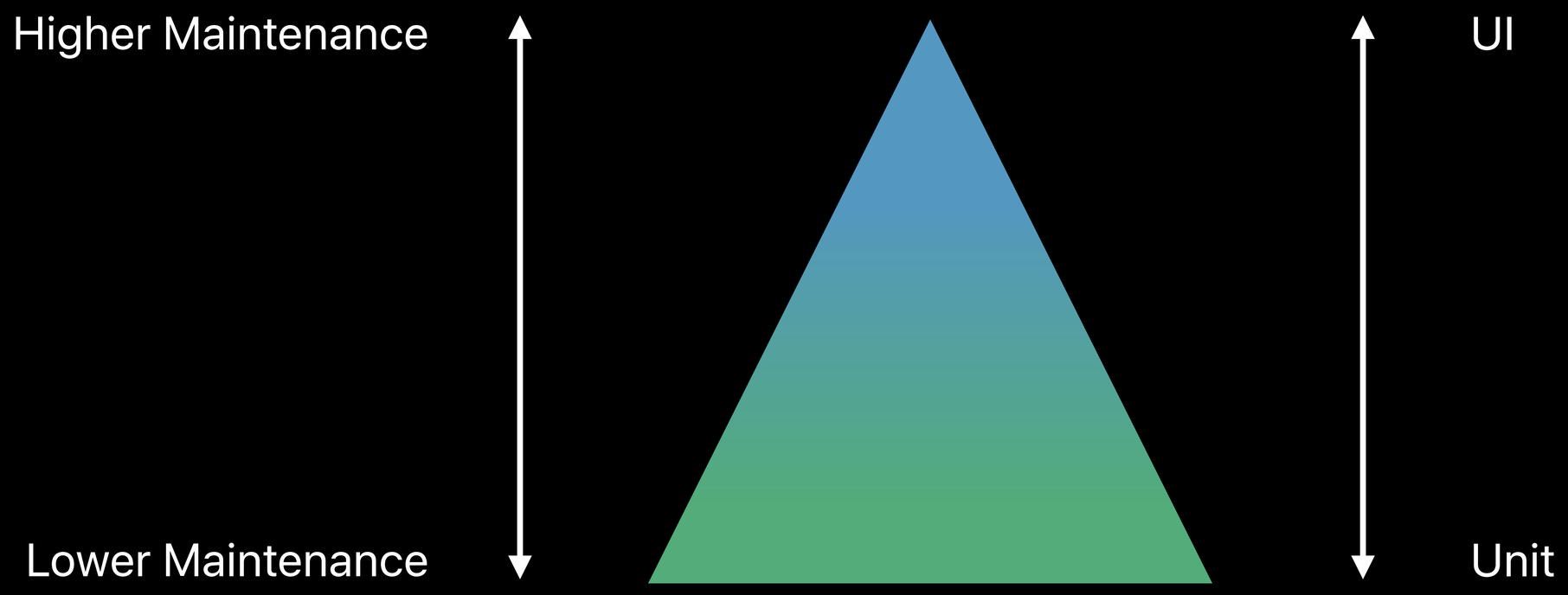
Testing Pyramid



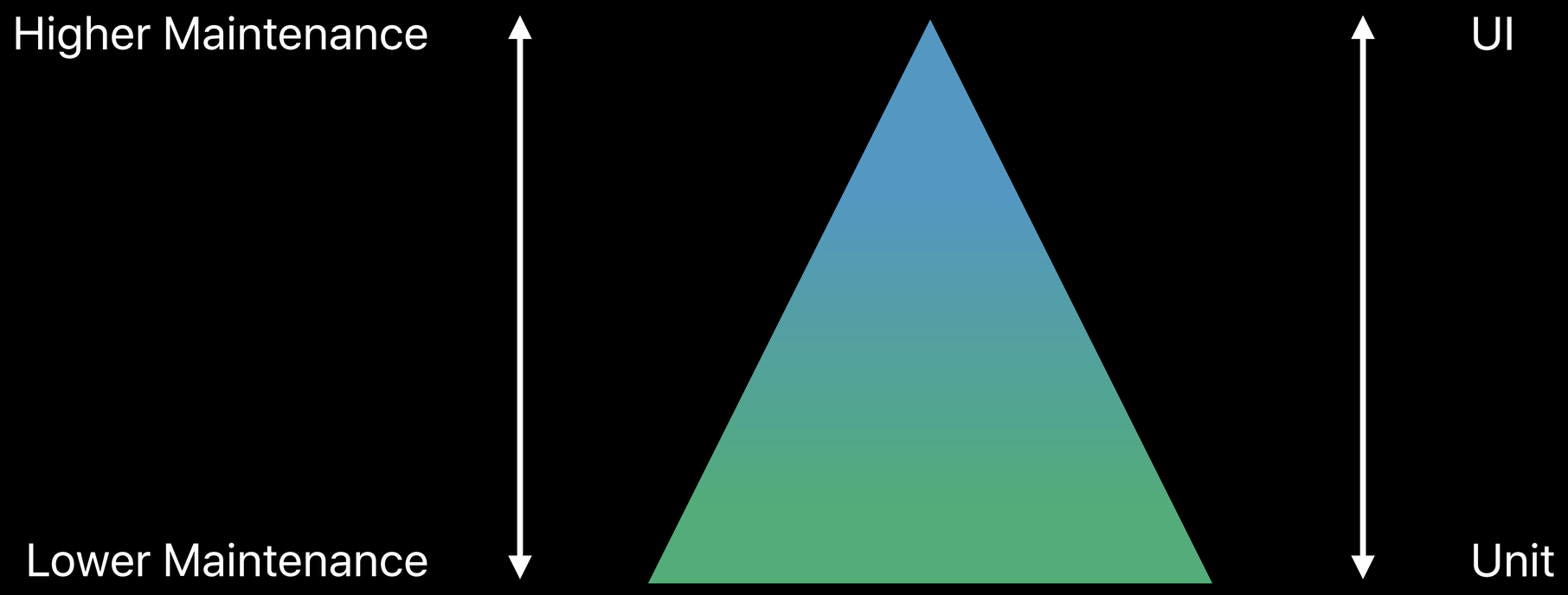
Testing Pyramid



Testing Pyramid



Testing Pyramid



Balance between UI and Unit Tests

Balance between UI and Unit Tests

Unit tests great for testing small, hard-to-reach code paths

Balance between UI and Unit Tests

Unit tests great for testing small, hard-to-reach code paths

UI tests are better at testing integration of larger pieces

Writing code to help UI tests scale

Code to Help UI Tests Scale

Code to Help UI Tests Scale

Abstracting UI element queries

Code to Help UI Tests Scale

Abstracting UI element queries

Creating objects and utility functions

Code to Help UI Tests Scale

Abstracting UI element queries

Creating objects and utility functions

Utilizing keyboard shortcuts

Code to Help UI Tests Scale

Abstracting UI element queries

Creating objects and utility functions

Utilizing keyboard shortcuts

```
app.buttons["blue"].tap()
app.buttons["red"].tap()
app.buttons["yellow"].tap()
app.buttons["green"].tap()
app.buttons["purple"].tap()
app.buttons["orange"].tap()
app.buttons["pink"].tap()
```

```
app.buttons["blue"].tap()
app.buttons["red"].tap()
app.buttons["yellow"].tap()
app.buttons["green"].tap()
app.buttons["purple"].tap()
app.buttons["orange"].tap()
app.buttons["pink"].tap()
```

```swift
func tapButton(_ color: String) {
    app.buttons[color].tap()
}
```

```swift
app.buttons["blue"].tap()
app.buttons["red"].tap()
app.buttons["yellow"].tap()
app.buttons["green"].tap()
app.buttons["purple"].tap()
app.buttons["orange"].tap()
app.buttons["pink"].tap()
```

```swift
func tapButton(_ color: String) {
    app.buttons[color].tap()
}
```

```swift
tapButton("blue")
tapButton("red")
tapButton("yellow")
tapButton("green")
tapButton("purple")
tapButton("orange")
tapButton("pink")
```

```
tapButton("blue")
tapButton("red")
tapButton("yellow")
tapButton("green")
tapButton("purple")
tapButton("orange")
tapButton("pink")
```

```
let colors = ["blue", "red", "yellow", "green", "purple", "orange", "pink"]
for color in colors {
    tapButton(color)
}
```

Abstracting UI Element Queries

Abstracting UI Element Queries

Store parts of queries in a variable

Abstracting UI Element Queries

Store parts of queries in a variable

Wrap complex queries in utility methods

Abstracting UI Element Queries

Store parts of queries in a variable

Wrap complex queries in utility methods

Reduces noise and clutter in UI test

Code to Help UI Tests Scale

Abstracting UI element queries

Creating objects and utility functions

Utilizing keyboard shortcuts

Code to Help UI Tests Scale

Abstracting UI element queries

Creating objects and utility functions

Utilizing keyboard shortcuts

```swift
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    app.buttons["Difficulty"].tap()
    app.buttons["beginner"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.buttons["Sound"].tap()
    app.buttons["off"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    app.buttons["Difficulty"].tap()
    app.buttons["beginner"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.buttons["Sound"].tap()
    app.buttons["off"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    app.buttons["Difficulty"].tap()
    app.buttons["beginner"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.buttons["Sound"].tap()
    app.buttons["off"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    app.buttons["Difficulty"].tap()
    app.buttons["beginner"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.buttons["Sound"].tap()
    app.buttons["off"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```
func setDifficulty(_ difficulty: String) {
    app.buttons["Difficulty"].tap()
    app.buttons[difficulty].tap()
    app.navigationBars.buttons["Back"].tap()
}
```

```swift
func setDifficulty(_ difficulty: String) {
    app.buttons["Difficulty"].tap()
    app.buttons[difficulty].tap()
    app.navigationBars.buttons["Back"].tap()
}

func setSound(_ sound: String) {
    app.buttons["Sound"].tap()
    app.buttons[sound].tap()
    app.navigationBars.buttons["Back"].tap()
}
```

```swift
func setDifficulty(_ difficulty: String) {
    // code
}

func setSound(_ sound: String) {
    // code
}
```

```swift
enum Difficulty {
    case beginner
    case intermediate
    case veteran
}

enum Sound {
    case on
    case off
}

func setDifficulty(_ difficulty: String) {
    // code
}

func setSound(_ sound: String) {
    // code
}
```

```swift
enum Difficulty {
    case beginner
    case intermediate
    case veteran
}

enum Sound {
    case on
    case off
}

func setDifficulty(_ difficulty: Difficulty) {
    // code
}

func setSound(_ sound: Sound) {
    // code
}
```

```
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    app.buttons["Difficulty"].tap()
    app.buttons["beginner"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.buttons["Sound"].tap()
    app.buttons["off"].tap()
    app.navigationBars.buttons["Back"].tap()
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    setDifficulty(.beginner)
    setSound(.off)
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    setDifficulty(.beginner)
    setSound(.off)
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```
class GameApp: XCUIApplication {

}
```

```
class GameApp: XCUIApplication {

    enum Difficulty { /* cases */ }
    enum Sound { /* cases */ }

}
```

```swift
class GameApp: XCUIApplication {

    enum Difficulty { /* cases */ }
    enum Sound { /* cases */ }

    func setDifficulty(_ difficulty: Difficulty) { /* code */ }
    func setSound(_ sound: Sound) { /* code */ }

}
```

```swift
class GameApp: XCUIApplication {

    enum Difficulty { /* cases */ }
    enum Sound { /* cases */ }

    func setDifficulty(_ difficulty: Difficulty) { /* code */ }
    func setSound(_ sound: Sound) { /* code */ }

    func configureSettings(difficulty: Difficulty, sound: Sound) { }

}
```

```swift
class GameApp: XCUIApplication {

    enum Difficulty { /* cases */ }
    enum Sound { /* cases */ }

    func setDifficulty(_ difficulty: Difficulty) { /* code */ }
    func setSound(_ sound: Sound) { /* code */ }

    func configureSettings(difficulty: Difficulty, sound: Sound) {
        app.navigationBars["Game.GameView"].buttons["Settings"].tap()
        setDifficulty(difficulty)
        setSound(sound)
        app.navigationBars.buttons["Back"].tap()
    }
}
```

```
func testGameWithDifficultyBeginnerAndSoundOff() {

    app.navigationBars["Game.GameView"].buttons["Settings"].tap()
    setDifficulty(.beginner)
    setSound(.off)
    app.navigationBars.buttons["Back"].tap()

    // test code

}
```

```swift
func testGameWithDifficultyBeginnerAndSoundOff() {

    GameApp().configureSettings(difficulty: .beginner, sound: .off)

    // test code

}
```

Creating Objects and Utility Functions

Creating Objects and Utility Functions

Encapsulate common testing workflows

Creating Objects and Utility Functions

Encapsulate common testing workflows

Cross-platform code sharing

Creating Objects and Utility Functions

Encapsulate common testing workflows

Cross-platform code sharing

Improves maintainability

```swift
class GameApp: XCUIApplication {

    enum Difficulty { /* cases */ }
    enum Sound { /* cases */ }

    func setDifficulty(_ difficulty: Difficulty) { /* code */ }
    func setSound(_ sound: Sound) { /* code */ }

    func configureSettings(difficulty: Difficulty, sound: Sound) {
        app.navigationBars["Game.GameView"].buttons["Settings"].tap()
        setDifficulty(difficulty)
        setSound(sound)
        app.navigationBars.buttons["Back"].tap()
    }
}
```

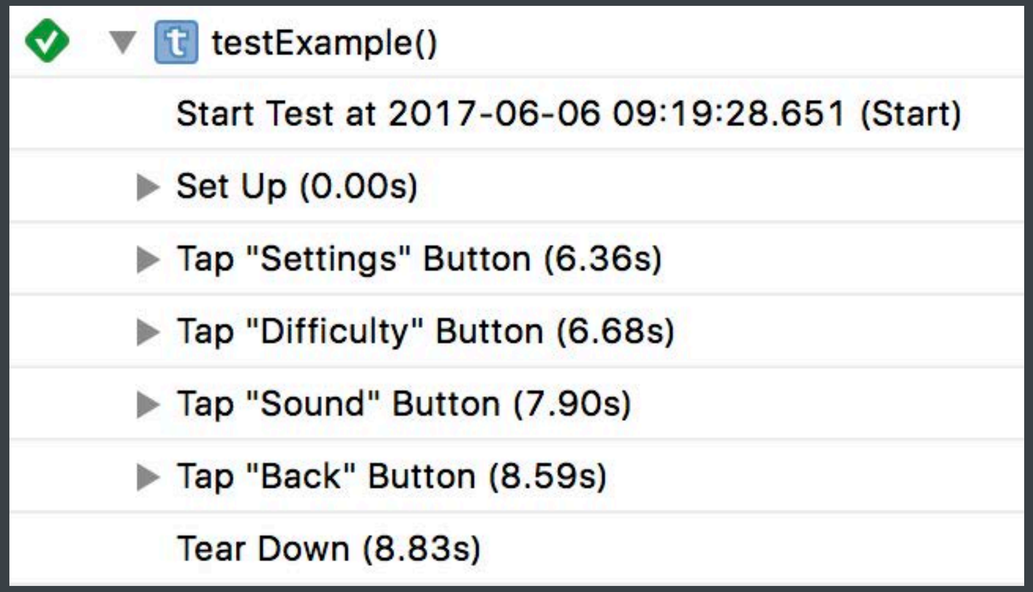
NEW

NEW

```
class GameApp: XCUIApplication {

    enum Difficulty { /* cases */ }
    enum Sound { /* cases */ }

    func setDifficulty(_ difficulty: Difficulty) { /* code */ }
    func setSound(_ sound: Sound) { /* code */ }

    func configureSettings(difficulty: Difficulty, sound: Sound) {
        XCTContext.runActivity(named: “Configure Settings: \(difficulty), \(sound)”) { _ in
            app.navigationBars["Game.GameView"].buttons["Settings"].tap()
            setDifficulty(difficulty)
            setSound(sound)
            app.navigationBars.buttons[“Back"].tap()
        }
    }
}
```

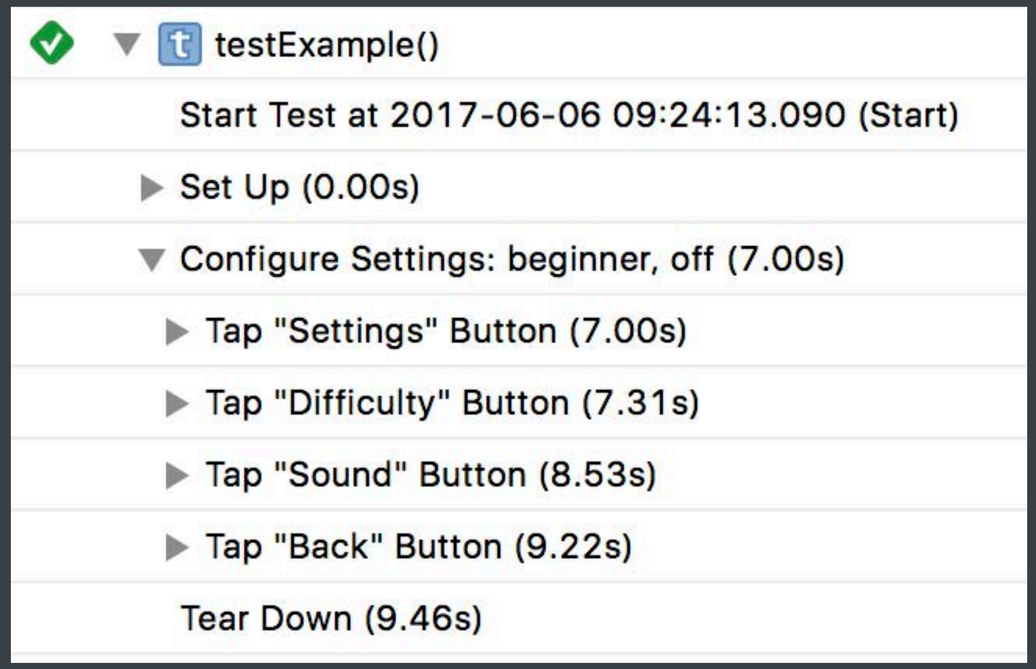
```
func testGameWithDifficultyBeginnerAndSoundOff() {

    GameApp().configureSettings(difficulty: .beginner, sound: .off)

    // test code

}
```

NEW



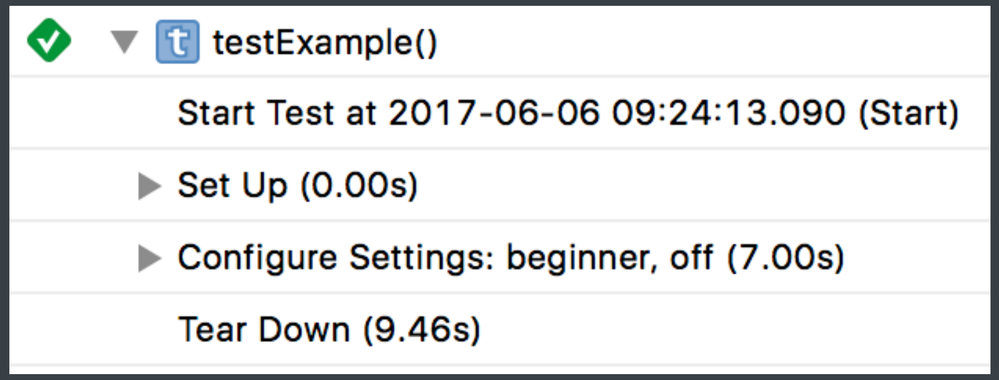
```
func testGameWithDifficultyBeginnerAndSoundOff() {

    GameApp().configureSettings(difficulty: .beginner, sound: .off)

    // test code

}
```

NEW



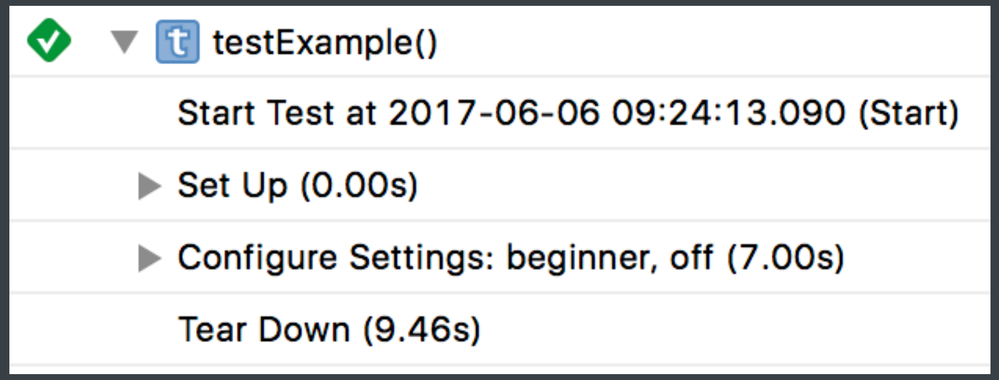
```
func testGameWithDifficultyBeginnerAndSoundOff() {

    GameApp().configureSettings(difficulty: .beginner, sound: .off)

    // test code

}
```

NEW



```
func testGameWithDifficultyBeginnerAndSoundOff() {

    GameApp().configureSettings(difficulty: .beginner, sound: .off)

    // test code

}
```

NEW



Code to Help UI Tests Scale

Abstracting UI element queries

Creating objects and utility functions

Utilizing keyboard shortcuts

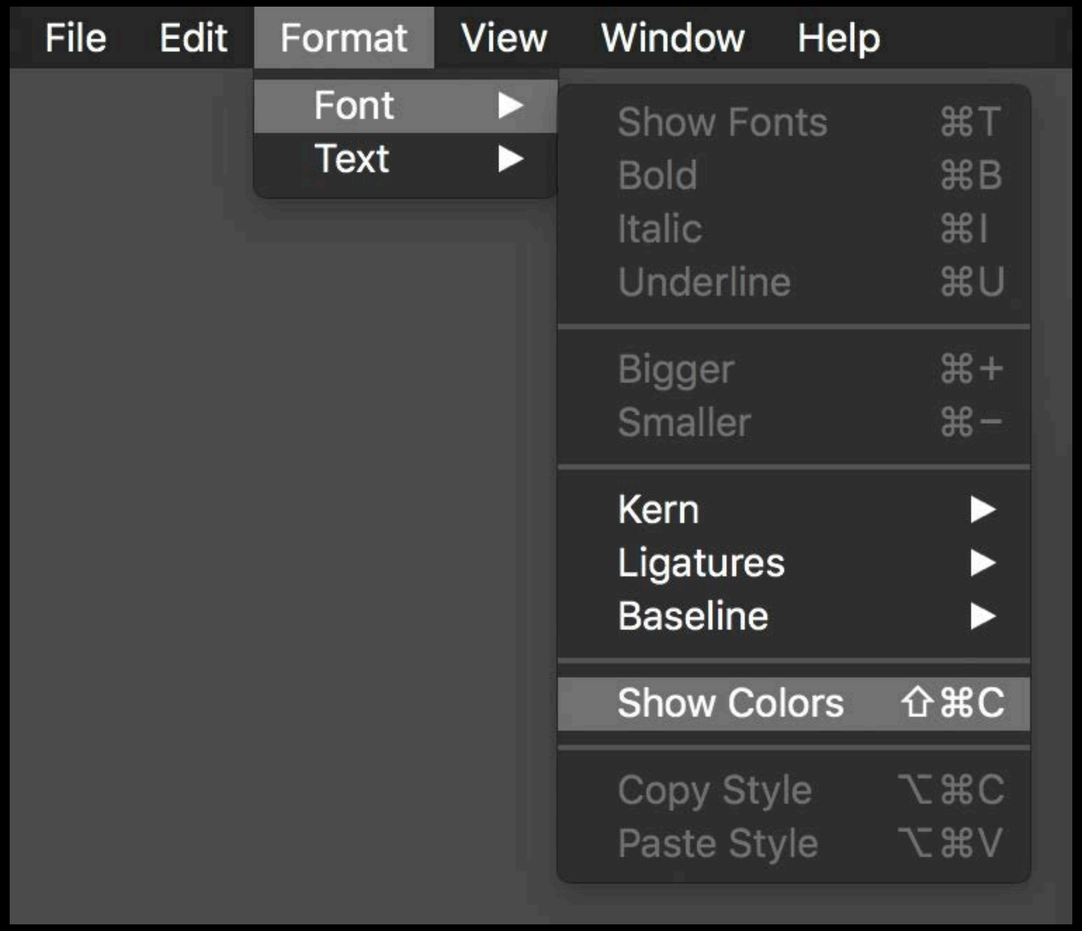
Code to Help UI Tests Scale

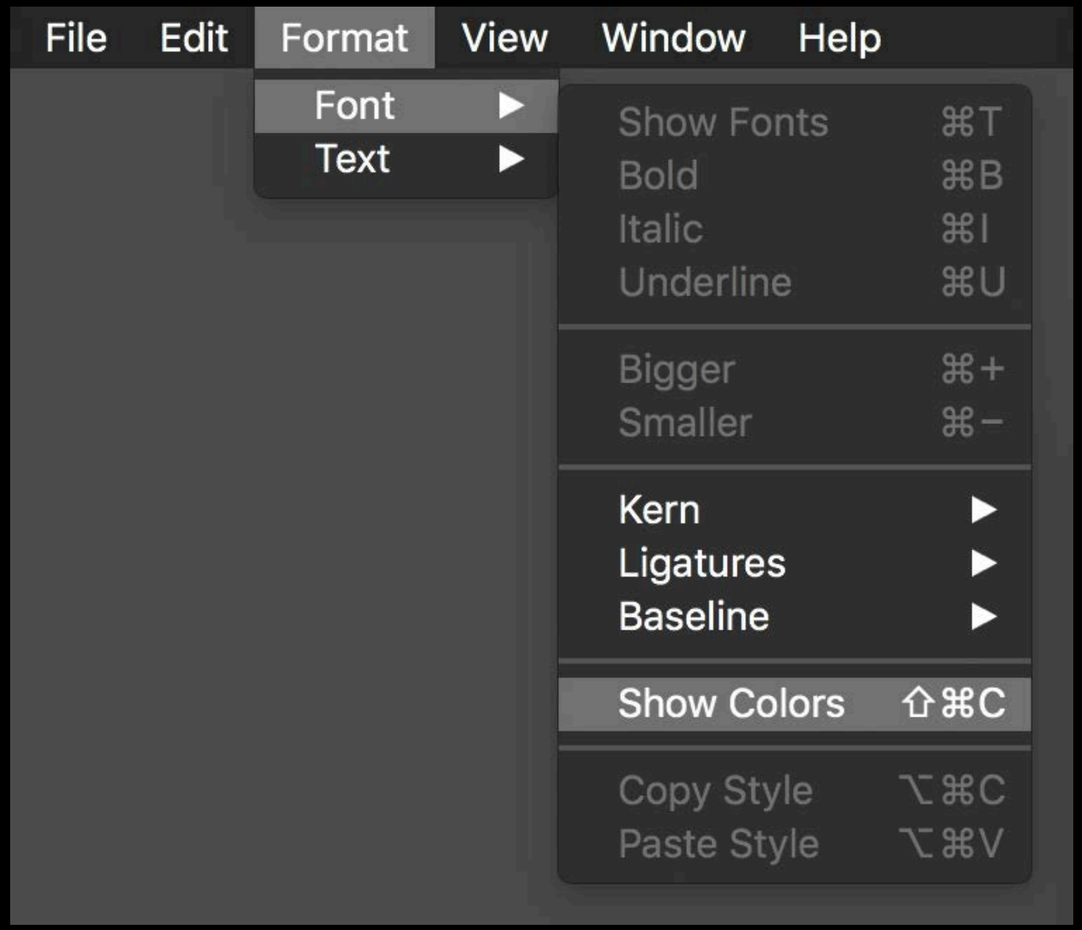
Abstracting UI element queries

Creating objects and utility functions

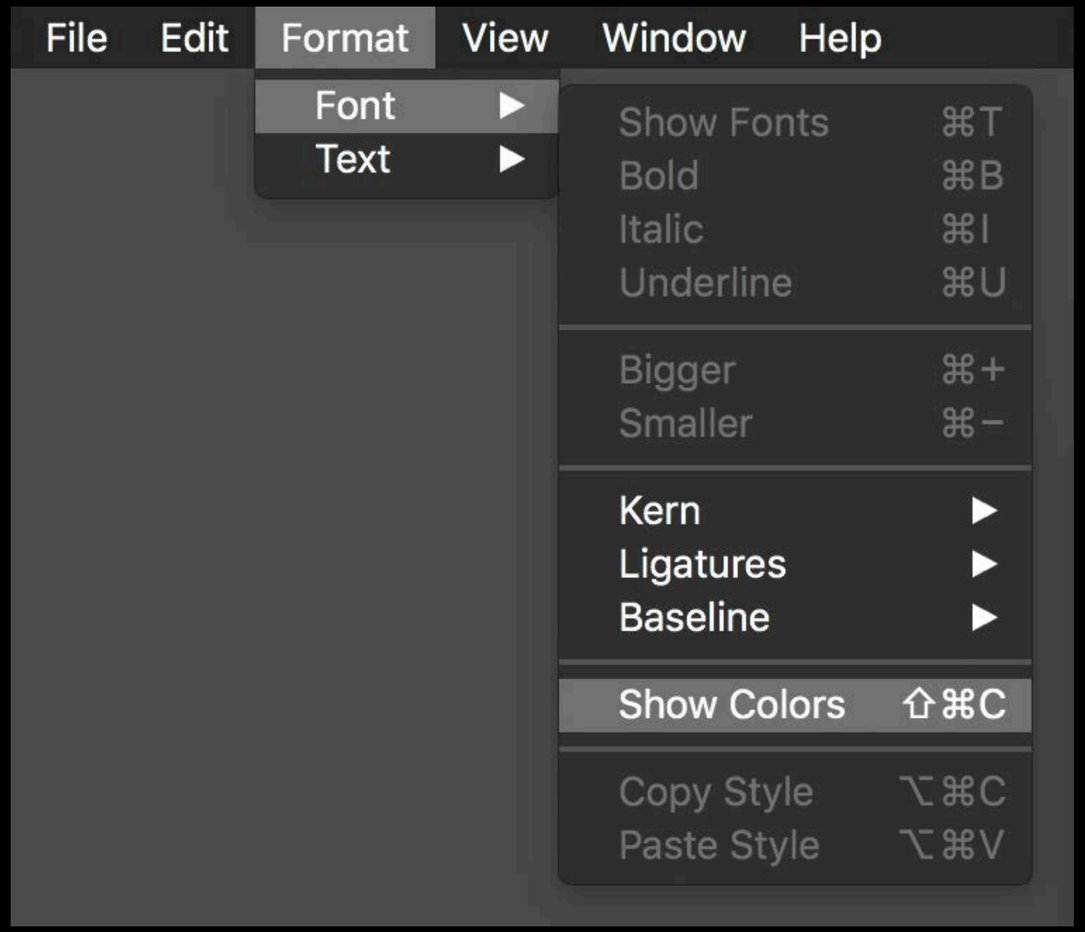
Utilizing keyboard shortcuts


Colors

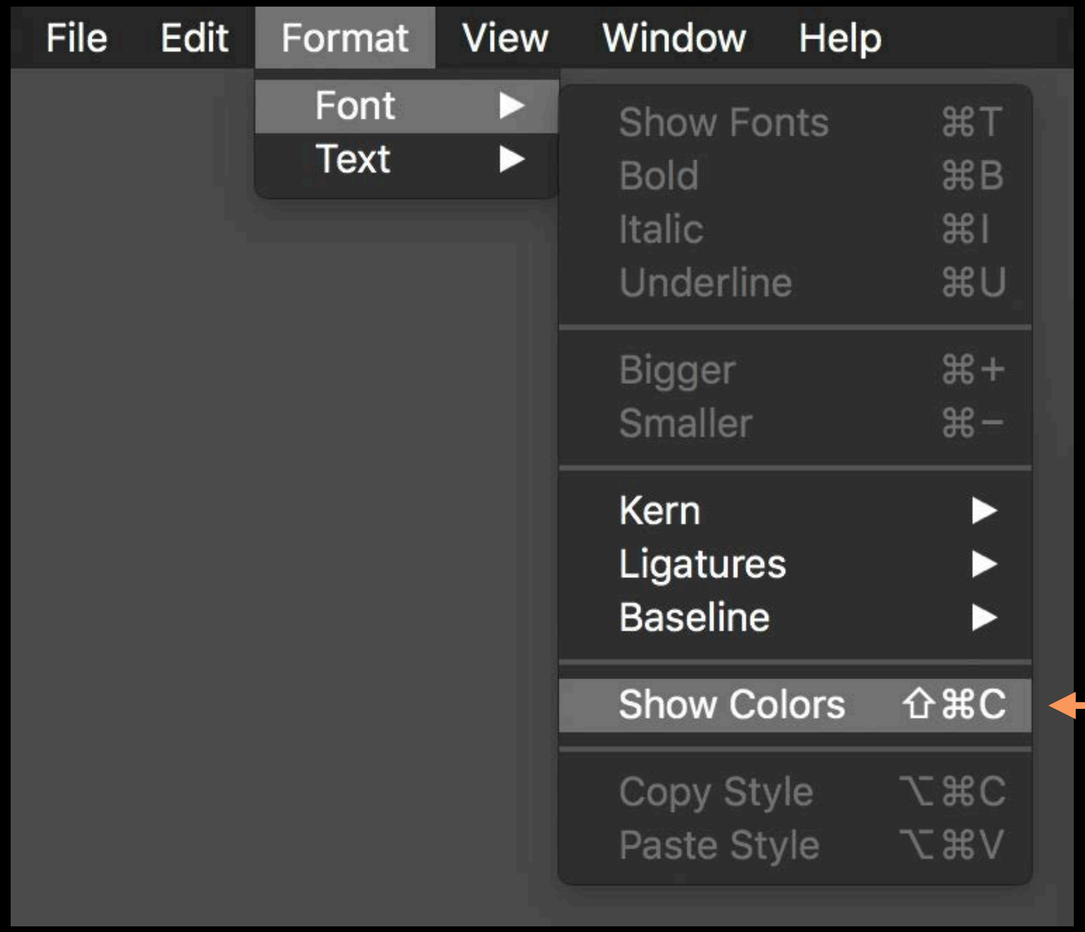

File
Edit
Format
View
Window
Help
Font
Text
Show Fonts ⌘T
Bold ⌘B
Italic ⌘I
Underline ⌘U
Bigger ⌘+
Smaller ⌘−
Kern
Ligatures
Baseline
Show Colors ⇧⌘C
Copy Style ⌥⌘C
Paste Style ⌥⌘V




```
let menuBarsQuery = app.menuBars
menuBarsQuery.menuBarItems["Format"].click()
menuBarsQuery.menuItems["Font"].click()
menuBarsQuery.menuItems["Show Colors"].click()
```




```
let menuBarsQuery = app.menuBars
menuBarsQuery.menuBarItems["Format"].click()
menuBarsQuery.menuItems["Font"].click()
menuBarsQuery.menuItems["Show Colors"].click()
```




```
let menuBarsQuery = app.menuBars
menuBarsQuery.menuBarItems["Format"].click()
menuBarsQuery.menuItems["Font"].click()
menuBarsQuery.menuItems["Show Colors"].click()
```

```
let menuBarsQuery = app.menuBars
menuBarsQuery.menuBarItems["Format"].click()
menuBarsQuery.menuItems["Font"].click()
menuBarsQuery.menuItems["Show Colors"].click()
```

```
let menuBarsQuery = app.menuBars
menuBarsQuery.menuBarItems["Format"].click()
menuBarsQuery.menuItems["Font"].click()
menuBarsQuery.menuItems["Show Colors"].click()
```

```
app.typeKey("c", modifierFlags:[.command, .shift])
```

```
func showColors() {
    app.typeKey("c", modifierFlags:[.command, .shift])
}
```

```
func testChangeColor() {

    let menuBarsQuery = app.menuBars
    menuBarsQuery.menuBarItems["Format"].click()
    menuBarsQuery.menuItems["Font"].click()
    menuBarsQuery.menuItems["Show Colors"].click()

    // test code

}
```

```
func testChangeColor() {
    showColors()
    // test code
}
```

Keyboard Shortcuts for UI Tests

Keyboard Shortcuts for UI Tests

Avoid working through menu bar for macOS

Keyboard Shortcuts for UI Tests

Avoid working through menu bar for macOS

Make code more compact

Treat your test code with the same amount of care as your app code

Quality of Test Code

Quality of Test Code

Important to consider even though it isn't shipping

Quality of Test Code

Important to consider even though it isn't shipping

Test code should support the evolution of your app

Quality of Test Code

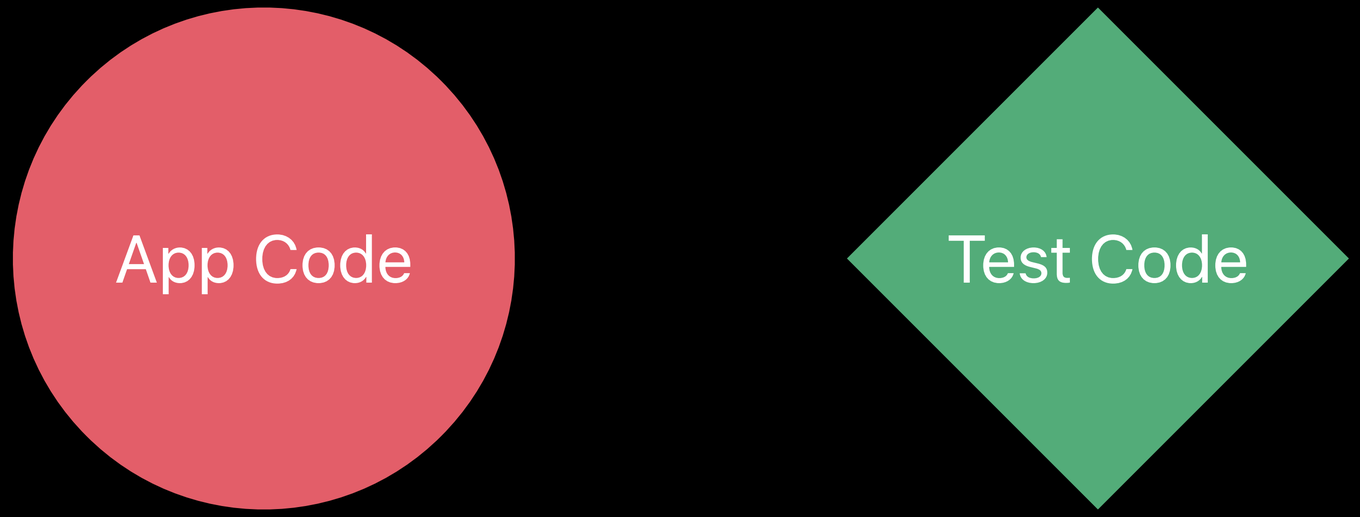
Important to consider even though it isn't shipping

Test code should support the evolution of your app

Coding principles in app code also apply to test code

Code reviews for test code,
not code reviews with test code

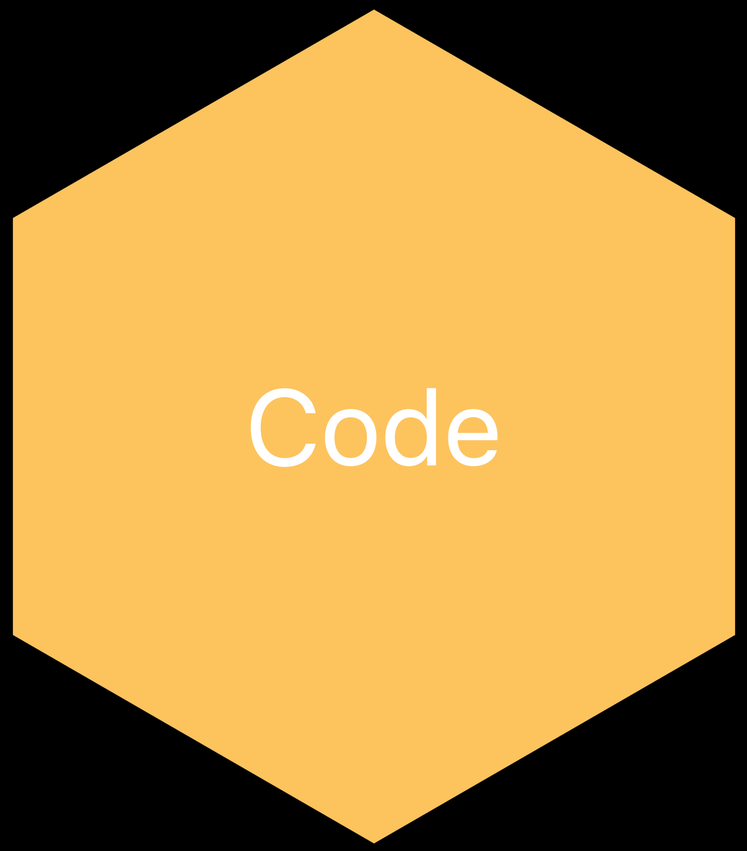
Quality of Test Code



Quality of Test Code

Quality of Test Code

Quality of Test Code



More Information

https://developer.apple.com/wwdc17/414

Related Sessions

What's New in Testing	WWDC 2017
Advanced Testing and Continuous Integration	WWDC 2016
UI Testing in Xcode	WWDC 2015
Testing in Xcode 6	WWDC 2014

Labs

Xcode Open Hours	Technology Lab K	Friday 1:50PM–4:00PM

WWDC17