|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Bug (SonarQube ref)** | **Code reference (file name, line no.)** | **Remarks** |
| 1 | Collection elements should not be replaced unconditionally **Noncompliant Code Example** letters["a"] = "Apple"  letters["a"] = "Boy" // Noncompliant  towns[i] = "London"  towns[i] = "Chicago" // Noncompliant |  |  |
| 2 | Collection sizes comparisons should make sense**Noncompliant Code Example** if (myArray.count >= 0) { ... }  if (myString.characters.count < 0) { ... } **Compliant Solution** if (myArray.isEmpty) { ... }  if (myString.isEmpty) { ... } |  |  |
| 3 | All branches in a conditional structure should not have exactly the same implementation**Noncompliant Code Example** if b == 0 { // Noncompliant  doOneMoreThing()  } else {  doOneMoreThing()  }  var b = a > 12 ? 4 : 4 // Noncompliant; always results in the same value  switch i { // Noncompliant  case 1:  doSomething()  case 2:  doSomething()  default:  doSomething()  } **Exceptions** This rule does not apply to if chains without else-s, or to switch-es without default clauses.  if b == 0 { // no issue, this could have been done on purpose to make the code more readable  doSomething()  } else if b == 1 {  doSomething()  } |  |  |
| 4 | Infix operators that end with "=" should update their left operands**Noncompliant Code Example** func \*\*= (p1:Int, p2:Int) -> Int { // Noncompliant. Change operator name or update value of first parameter  return p1 \*\* p2  }  func => (p1:Int, p2:Int) -> Int { // Compliant; doesn't end with '='  return p1 \*\* p1 \*\* p2  } **Compliant Solution** func \*\*= (inout p1:Int, p2:Int) {  p1 = p1 \*\* p2  }  func => (p1:Int, p2:Int) -> Int {  return p1 \*\* p1 \*\* p2  } |  |  |
| 5 | Precedence and associativity of standard operators should not be changed**Noncompliant Code Example** infix operator - : CustomAdditionPrecedence // Noncompliant. For a different behavior create a different operator  precedencegroup CustomAdditionPrecedence {  associativity: right  }  func - (lhs: MyInt, rhs: MyInt) -> MyInt {  // ...  }  var a = MyInt(10), b = MyInt(5), c = MyInt(5)  print(a - b - c) // against expectations, this outputs 10 **Compliant Solution** infix operator <- : CustomAdditionPrecedence  precedencegroup CustomAdditionPrecedence {  associativity: right  }  func <- (lhs: MyInt, rhs: MyInt) -> MyInt {  // ...  }  var a = MyInt(10), b = MyInt(5), c = MyInt(5)  var a = MyInt(10), b = MyInt(5), c = MyInt(5)  print(a - b - c) // prints 0 as expected  print(a <- b <- c) // prints 10 |  |  |
| 6 | Return values from functions without side effects should not be ignored |  |  |
| 7 | Related "if/else if" statements and "cases" in a "switch" should not have the same condition**Noncompliant Code Example** if param == 1 {  openWindow()  } else if param == 2 {  closeWindow()  } else if param == 1 { // Noncompliant  moveWindowToTheBackground()  }  switch i {  case 1:  //...  case 3:  //...  case 1: // Noncompliant  //...  default:  // ...  } **Compliant Solution** if param == 1 {  openWindow()  } else if param == 2 {  closeWindow()  } else if param == 3 {  moveWindowToTheBackground()  }  switch i {  case 1:  //...  case 3:  //...  default:  // ...  } |  |  |
| 8 | Identical expressions should not be used on both sides of a binary operator**Noncompliant Code Example** if a == a { // always true  doZ()  }  if a != a { // always false  doY()  }  if a == b && a == b { // if the first one is true, the second one is too  doX()  }  if a == b || a == b { // if the first one is true, the second one is too  doW()  }  var j = 5 / 5 //always 1  var k = 5 - 5 //always 0 **Exceptions** Left-shifting 1 onto 1 is common in the construction of bit masks, and is ignored.  var i = 1 << 1; // Compliant  var j = a << a; // Noncompliant |  |  |
| 9 | All code should be reachable**Noncompliant Code Example** func fun(a:Int)->Int{  var i = 10;  return i + a;  i++; // this is never executed  } **Compliant Solution** func fun(a:Int)->Int{  var i = 10;  return i + a;  } |  |  |
| 10 | Loops with at most one iteration should be refactored**Noncompliant Code Example** for i in 1...10 { // noncompliant, loop only executes once  print("i is \(i)")  break  } **Compliant Solution** for i in 1...10 {  print("i is \(i)")  } |  |  |
| 11 | Implicitly unwrapped optionals should not be used**Noncompliant Code Example** var greeting : String! // Noncompliant  println(greeting) // At this point the value is nil. Runtime error results **Compliant Solution** var greeting : String?  if let howdy = greeting {  println(howdy)  } |  |  |
| 12 | The ternary operator should not return the same value regardless of the condition**Noncompliant Code Example** func canVote(person:Person) -> Bool {  return person.age > 18 ? true : true // Noncompliant; is this what was intended?  } **Compliant Solution** func canVote(person:Person) -> Bool {  return person.age > 18 ? true : false  } |  |  |
| 13 | Floating point numbers should not be tested for equality**Noncompliant Code Example** var myNumber: Float = 0.3 + 0.6  if myNumber == 0.9 { // Noncompliant. Because of floating point imprecision, this will be false  // ...  }  if myNumber <= 0.9 && myNumber >= 0.9 { // Noncompliant indirect equality test  // ...  }  if myNumber < 0.9 || myNumber > 0.9 { // Noncompliant indirect inequality test  // ...  } |  |  |
| 14 | Useless "if true {...}" and "if false {...}" blocks should be removed**Noncompliant Code Example** if true { // Noncompliant  doSomething()  }  ...  if false { // Noncompliant  doSomethingElse()  } **Compliant Solution** doSomething() |  |  |