
STRANGELOVE
OR:
HOW I
LEARNED
TO STOP
WORRYING
AND
LOVE THE
BOMB
U.S. AIR FORCE
U.S. AIR FORCE
COPYRIGHT (C) MCMLXIII HAWK FILMS LTD. ALL RIGHTS RESERVED

Brent Royal-Gordon

TWITTER, GITHUB, ETC.

@brentdax

Swift is what you'll write apps in for the next fifteen years

```
class ViewController: UIViewController {
  override func viewDidAppear(animated: Bool) {
    super.viewDidAppear(animated)

    let alert = UIAlertView(
        title: "Hello world!",
        message: nil,
        delegate: nil,
        cancelButtonTitle: "Hi!")
    alert.show()
  }
}
```

```
class
  override


        title:
        message:
        delegate:
        cancelButtonTitle:
    alert.
  }
}
```

```
class ViewController: UIViewController {
  override func viewDidAppear(animated: Bool) {
    super.viewDidAppear(animated)

    let alert = UIAlertView(
        title: "Hello world!",
        message: nil,
        delegate: nil,
        cancelButtonTitle: "Hi!")
    alert.show()
  }
}
```

What’s forbidden

Why it’s forbidden

What you should do about it

No Implicit Conversions

```
- (NSInteger)numberOfRows {
    NSUInteger count = …;
    return count;
}
```

```
- (   Integer
      UInteger count
    return count
}
```

```
func numberOfRows() -> Int {
    let count: UInt = …
    return count
}
```

‘UInt’ is not convertible to ‘Int’

```
func numberOfRows() -> Int {
    let count: UInt = …
    return Int(count)
}
```

```
- (BOOL)hasObjects {
    return self.objects.count;
}
```

```
container.objects = arrayWith256Objects;

NSAssert(container.hasObjects,
    "This can’t fail…right? Right?");
```

```swift
// Before
let NUM_COLUMNS: Int = 4
gridSize.width *= CGFloat(NUM_COLUMNS)

// After
var NUM_COLUMNS: CGFloat = 4
gridSize.width *= NUM_COLUMNS
```

```
…
var gridSize = cellSize

gridSize.width *= CGFloat(NUM_COLUMNS)
gridSize.height *=
    CGFloat(items.count / NUM_COLUMNS)
…
```

```
extension CGSize {
    func sizeOfCells(count: Int,
      columns: Int = 1) -> CGSize {
        return CGSize(
            width: width  * CGFloat(columns),
           height: height * CGFloat(count / columns)
        )
    }
}
```

```
…
let gridSize = cellSize.sizeOfCells(items.count,
        columns: NUM_COLUMNS)
…
```

```
func *= (inout lhs: CGFloat, rhs: Int) {
    lhs *= CGFloat(rhs)
}

gridSize.width *= NUM_COLUMNS
gridSize.height *= items.count / NUM_COLUMNS
```

```
private func *= (inout lhs: CGFloat, rhs: Int) {
    lhs *= CGFloat(rhs)
}

gridSize.width *= NUM_COLUMNS
gridSize.height *= items.count / NUM_COLUMNS
```

II

Swift Optionals

```
NSView * myView = nil;
```

```
var myView: NSView = nil
```

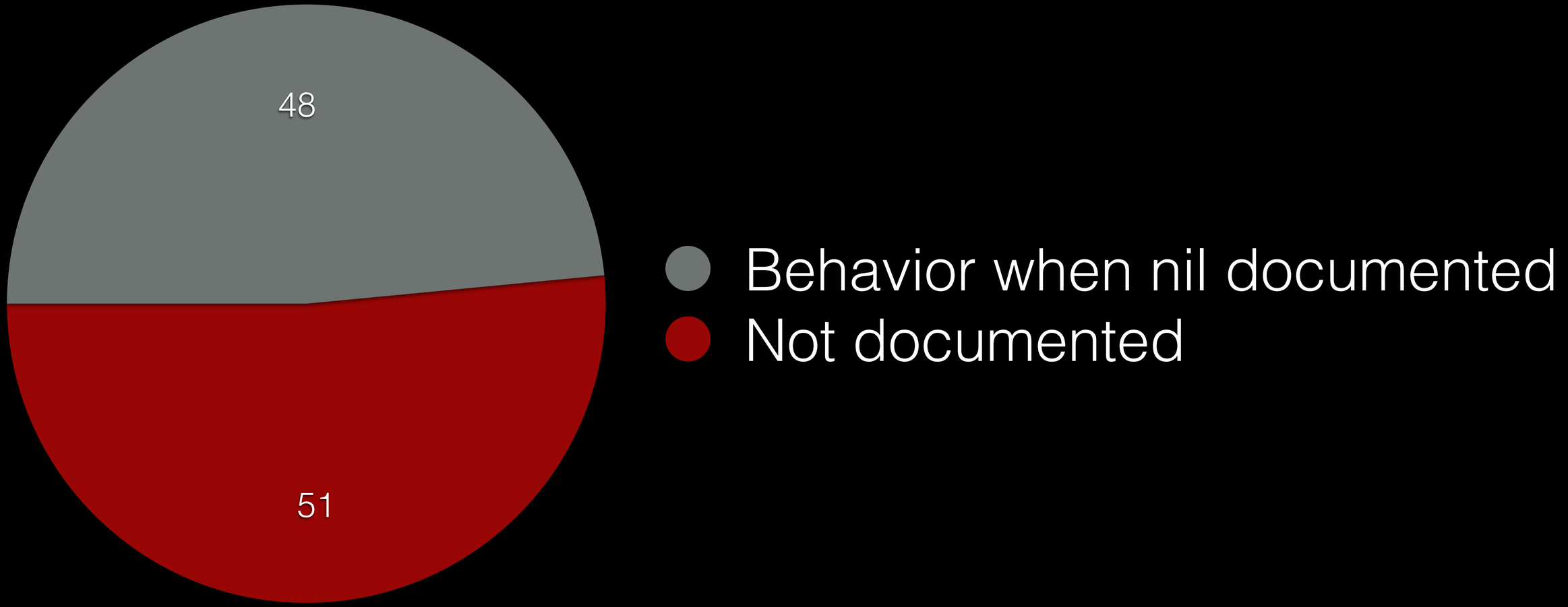
type 'NSView' does not conform to protocol 'NilLiteralConvertible'

```
self.label.stringValue =
    @"This text doesn’t appear!";
```

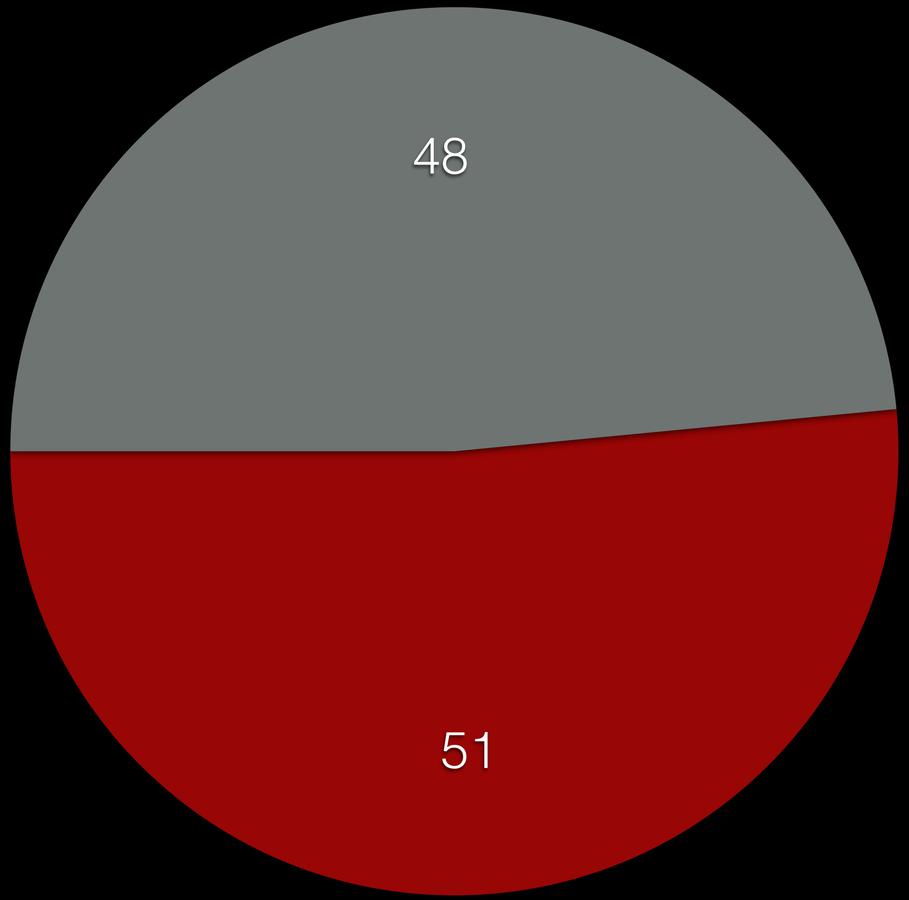
```
NSData * data = [NSData dataWithContentsOfURL:url
    options:0 error:&error];
if(!data) {
    // TODO: handle error
}
[self doSomethingWithBuffer:data.bytes];
```

```
if([myString hasPrefix:possiblyNilString]) {
```

Documentation of `nil` parameters



Documentation of `nil` parameters in `NSString`



- Behavior when nil documented
- Not documented

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965…. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

– Tony Hoare, designer of the ALGOL type system
(and discoverer of Quicksort)

```
var myView: NSView? = nil
```

```
var myIndex: Int? = find(array, value)

if myIndex == nil {
    return
}
```

```
var myInt: Int? = nil

assert(myInt != 0)
assert(myInt != -1)
assert(myInt != NSNotFound)

assert(myInt == nil)
```

```
aView.addSubview( optionalView )
```

value of optional type 'NSView?' not unwrapped

```
aView.addSubview( optionalView! )
```

```
let image = NSImage(named: "PlayIcon")!
```

```
aView.addSubview( optionalView ?? defaultView )
```

```
if let existingView = optionalView {
    aView.addSubview( existingView )
}
```

```
optionalView?.removeFromSuperview()
```

```swift
// Before you write:
var myString: String? = nil

// Consider using:
var myString: String = ""
```

```
- (instancetype)initWithURL:(NSURL*)URL {
    if((self = [super init])) {
        NSDictionary * plist = [NSDictionary
            dictionaryWithContentsOfURL:URL];
        if(!plist) {
            return nil;
        }
        …
    }
    return self;
}
```

```
init?(URL: NSURL) {
    super.init()
    let plist = NSDictionary(contentsOfURL: URL)
    if plist == nil {
        return nil
    }
    …
}
```

III

No Type Changes

```
@interface SunsetView: NSView

@property CAGradientLayer * layer;

@end
```

```
class SunsetView: NSView {
    override var layer: CAGradientLayer? {
        get {
            return super.layer as CAGradientLayer?
        }
        set (newValue) {
            super.layer = newValue
        }
    }
```

Cannot override mutable property ‘layer’ of type ‘CALayer?’ with covariant type ‘CAGradientLayer?’


Shoot it off! Shoot with the gun!

```
class SunsetView: NSView {
    override var layer: CAGradientLayer? {
        get {
            return super.layer as CAGradientLayer?
        }
        set (newValue) {
            super.layer = newValue
        }
    }
```

Cannot override mutable property ‘layer’ of type ‘CALayer?’ with covariant type ‘CAGradientLayer?’

```
let sunsetView = SunsetView(frame: self.bounds)
let aView = sunsetView as NSView
aView.layer = CALayer()

assert(sunsetView.layer is CAGradientLayer,
    "This can’t fail, right? Right?")
```

Liskov Substitution Principle

“Let *q(x)* be a property provable about objects *x* of type *T*.

“Then *q(y)* should be provable for objects *y* of type *S*, where *S* is a subtype of *T*.”

```
class MyClass {
    func doIt(param: NSResponder) -> NSResponder
}

class MySubclass: MyClass {
    override func doIt(param: NSObject) -> NSView
}
```

```
class
                    NSResponder  -> NSResponder
}

class
                        NSObject  -> NSView
}
```

```
class NSResponder  -> NSResponder
{
}

class NSView  -> NSObject
{
}
```

Method does not override any method from its superclass
(this is considered an overload instead, and doesn’t take the “override” keyword)

```
class SunsetView: NSView {
    override var layer: CAGradientLayer? {
        get {
            return super.layer as CAGradientLayer?
        }
        set (newValue) {
            super.layer = newValue
        }
    }
```

Cannot override mutable property ‘layer’ of type ‘CALayer?’ with covariant type ‘CAGradientLayer?’

```
class SunsetView: NSView {
    var gradientLayer: CAGradientLayer? {
        get {
            return super.layer as CAGradientLayer?
        }
        set (newValue) {
            super.layer = newValue
        }
    }
```

IV

Safe Initialization

```
class Person {
    var name: String

    init() { /* do nothing */ }
}
```

Property 'self.name' not initialized

```
class Person: NSObject {
    var name: String

    override init() {
        super.init()
        self.name = "Jack"
    }
}
```

Property 'self.name' not initialized at super.init call

```
class Person: NSManagedObject {
    override init(entity: NSEntityDescription,
      insertIntoManagedObjectContext:
      NSManagedObjectContext?) {
        // some initialization
    }
}
```

super.init isn't called before returning from initializer

```
class PersonDocument: NSDocument {
    init(type: String, error: NSErrorPointer) {
        super.init(type: type, error: error)
    }
}
```

must call a designated initializer of the superclass 'NSDocument'

```
class Person {
    var name: String
    init?(URL: NSURL) {
        if let plist = NSDictionary(contentsOfURL: URL) {
            name = plist["name"] as String
        }
        else {
            return nil
        }
```

all stored properties of a class instance must be initialized before returning nil from an initializer

bit.ly/DrSwiftlove

```objc
@interface NPKUndoableObject: NSObject

// Override these
@property (readonly) NSUndoManager * undoManager;
+ (NSSet*)keyPathsForUndoRegistration;

// You must call these or bad things will happen
- (instancetype)init;
- (void)dealloc;

@end
```

```
- (              init {


          [[
            [self addObserver:self forKeyPath:keyPath
               options:NSKeyValueObservingOptionPrior
               context:KVO];
        }
    }

}
```

```
- (      dealloc {
    [
    
    
    
      [[
        [self removeObserver:self forKeyPath:keyPath
           context:KVO];
    }
}
```

```objc
@implementation Person

- (instancetype)initWithURL:(NSURL*)URL {
    NSDictionary * plist = [NSDictionary
        dictionaryWithContentsOfURL:URL];
    if(!plist) {
        return nil;
    }
    if((self = [super init])) {
        _name = plist[@"name"];
```

```
class Person {
    var name: String
    var administrator: Bool = false
    var birthdate: NSDate?

    init(name: String) {
        self.name = name
    }
}
```

```
init(coder: NSCoder?) {
    // Lots of complicated initialization
    super.init(coder: coder)
}

init(frame: NSRect?) {
    // Identical complicated initialization
    super.init(frame: frame)
}
```

```
init(coder: NSCoder?) {
    (name, age) = commonInit()
    super.init(coder: coder)
}

init(frame: NSRect?) {
    (name, age) = commonInit()
    super.init(frame: frame)
}
```

```
init(coder: NSCoder?) {
    (name, age) = Person.commonInit()
    super.init(coder: coder)
}

init(frame: NSRect?) {
    (name, age) = Person.commonInit()
    super.init(frame: frame)
}
```

```
init(coder: NSCoder?, orFrame: CGRect?) {
    // Lots of complicated initialization
    if let coder = coder {
        super.init(coder: coder)
    }
    else {
        super.init(frame: frame!)
    }
}
```

v

Required Initializers

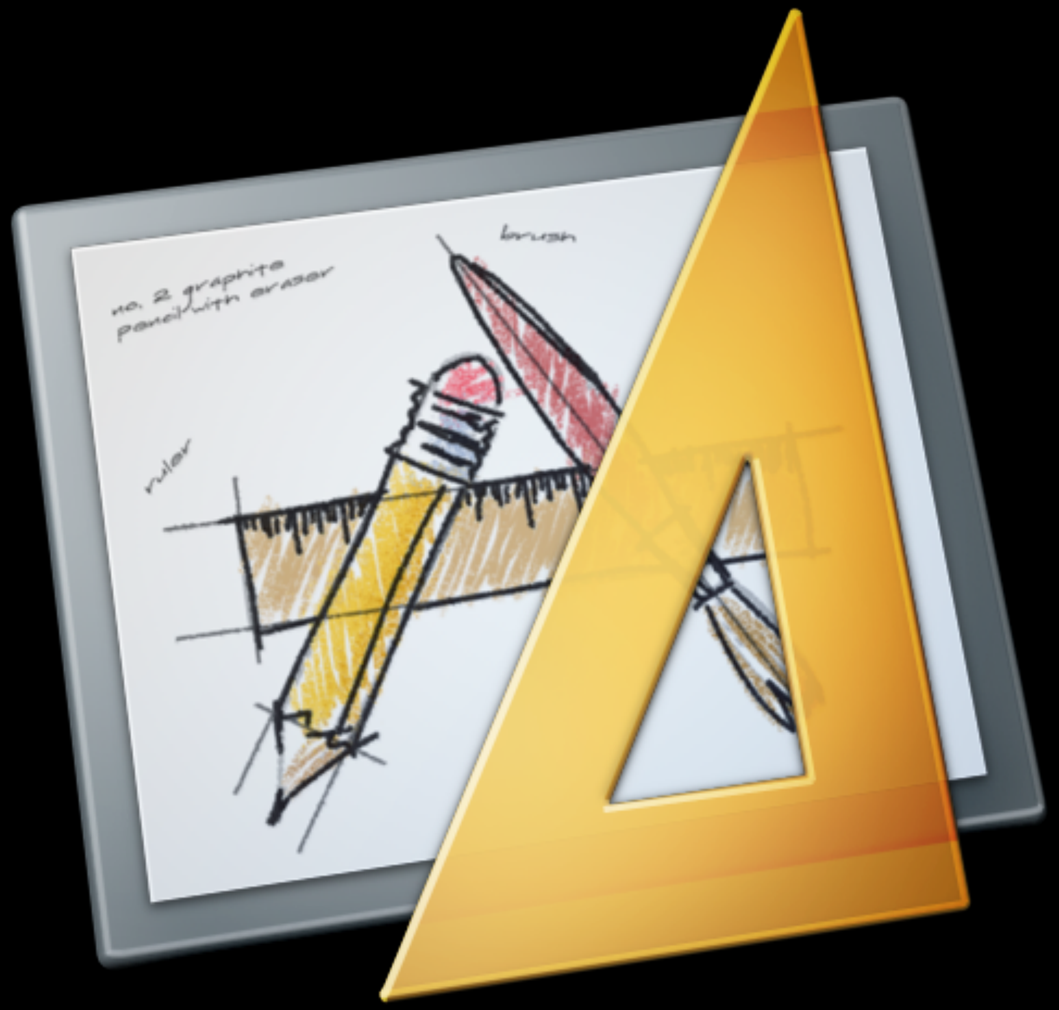
```
protocol Typeable {
    init?(text: String)
}
class IDNumber: Typeable { … }
class SocialSecurityNumber: IDNumber {
    init() { … }
    required init?(text: String) {
       super.init(text: text)
    }
}
```

```
class MyView: NSView {
    init() { … }
    …
}
```

‘required’ initializer ‘init(coder:)’ must be provided by subclass of ‘NSView’

```
@implementation MyTextField

- (id)initWithFrame:(CGRect)frame {
    if((self = [super initWithFrame:frame])) {
        self.textColor = [NSColor blueColor];
    }
    return self;
}
```


no. 2 graphite
pencil with eraser
brush
ruler


Epic fail


WORLD TARGETS IN MEGADEATHS
WAR ALERT ACTIONS BOOK

```
class PageStackView: NSView {
    // Instead of:
    var numberOfPages: Int
    override init(frame: NSRect) {
        numberOfPages = 1
        super.init(frame: frame)
    }

    // Just do:
    var numberOfPages: Int = 1
}
```

```
required init(coder: NSCoder) {
    …
}
```

```
required init(coder: NSCoder) {
    fatalError(
        "init(coder:) has not been implemented")
}
```

No Implicit Conversions

Swift Optionals

No Type Changes

Safe Initialization

Required Initializers

mmutable Value Types Constants Everywhere Cast-Or-Return-

pe-Safe Enumerations No Implicit Conversions Type-Safe Collections

es With Associated Values Swift Optionals `inout`

ng No Type Changes Incorrect Casts

Raw Pointers Rarely Used Safe Initialization Auto-Copying Value Type

Substitution Forbidden Required Initializers Convenience Initializers

ways Uses ARC Unsafe Operations Clearly Marked Protocols Everyw

Writing: a little slower

Debugging: way faster

Less boilerplate, more meaning

Stop worrying and love the error.

THE END

TALK GIVEN BY

BRENT ROYAL-GORDON

@BRENTDAX

AT THE 2014 MACTECH CONFERENCE

4:30 TOMORROW

Swift Lab