ThoughtWorks®

SWIFT 2.0

What's new, what's better?

Hermann Vocke - ***@hamvocke***

This talk will be about the novelties introduced with Swift 2.0. While we'll mostly focus on what's new in Swift 2.0 we will of course also explore some of the basic language features of Swift. So if you're not yet familiar with Swift at all this is not a problem, just tell me to slow down whenever you feel like you need further explanation.

Quick Check:

WHO OF YOU HAS ALREADY WRITTEN SOME SWIFT CODE?

Before we start I’d love to know who of you already has written some Swift code before — professionally, as a hobby project, as part of a tutorial, doesn’t matter

That's me,

HAM

Software Developer, Consultant, ThoughtWorker, Meetup Organiser, Tinkerer, Maniac, Dad

@hamvocke www.hamvocke.com

Who am I and what qualifies me to stand here and tell you stuff about Swift 2.0?

I'm Ham, Software Developer at ThoughtWorks and I've been working on a Swift project for the past 12 months. We started with Swift 1.0, went through all the hoops and headaches and are now on the step of migrating to Swift 2.0.

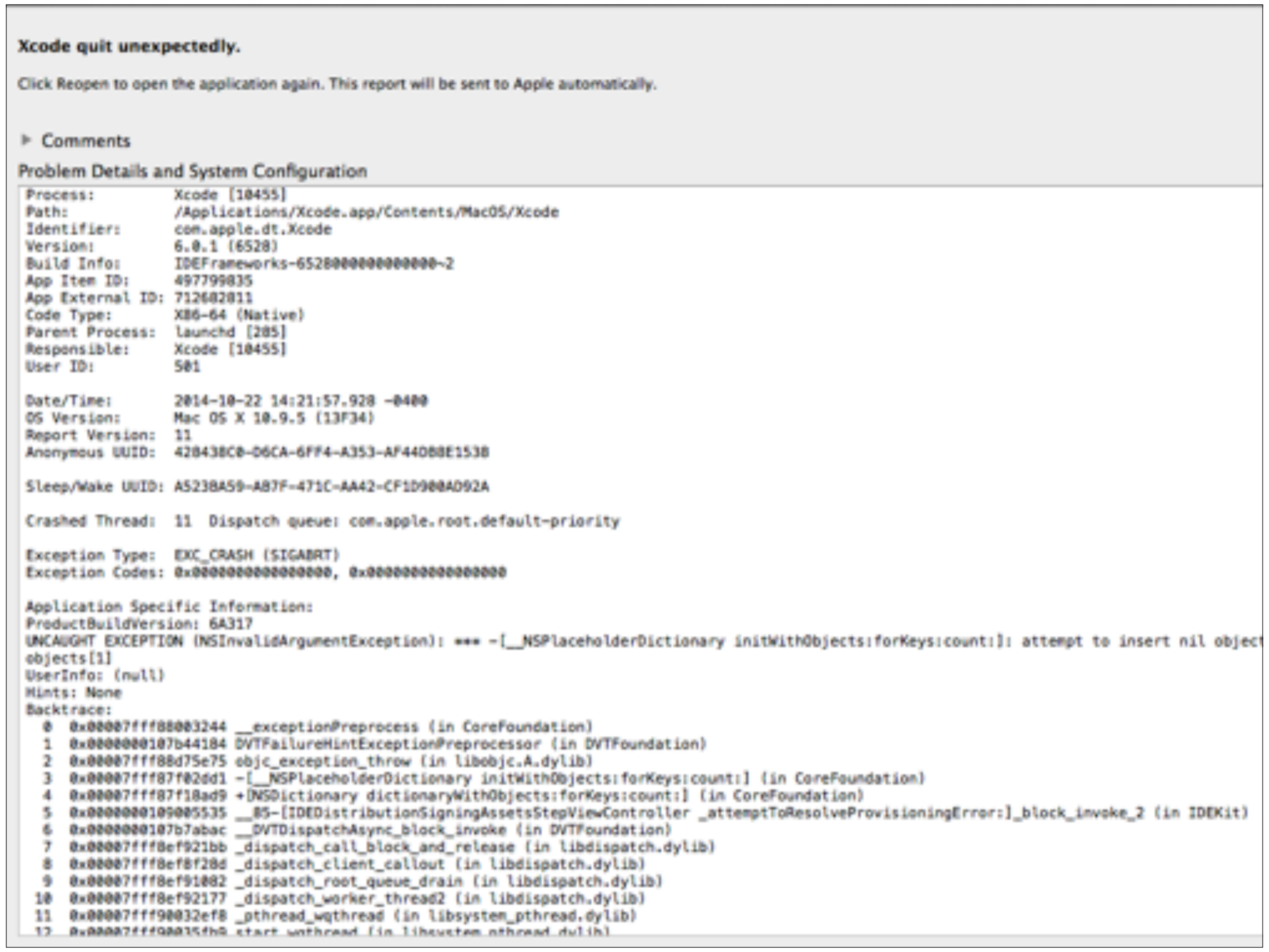
I'm certainly not the most knowledgable Swift developer around, perhaps not even in this room. But I still think that I can show you some bits and pieces of the latest Swift upgrade. I've never been an Objective-C dev so I might lack some background here and there. If there's someone around who has more background in the history of iOS development and feels that I'm telling you silly stuff, just feel free to let me know immediately so that we can all learn a lot from this.

BACK TO 2014

Before we look into Swift 2.0 let’s take a look back to what happened in 2014

As most of you know, Apple announced Swift as their new programming language at WWDC 2014. It was announced to be the way to develop iOS and Mac OS apps in the future as a replacement for Objective-C.

Apple made sure that Swift was fully compatible with Objective-C code and can live and exist in the same codebase — also known as “Mix and Match” approach.



Xcode quit unexpectedly.

Click Reopen to open the application again. This report will be sent to Apple automatically.

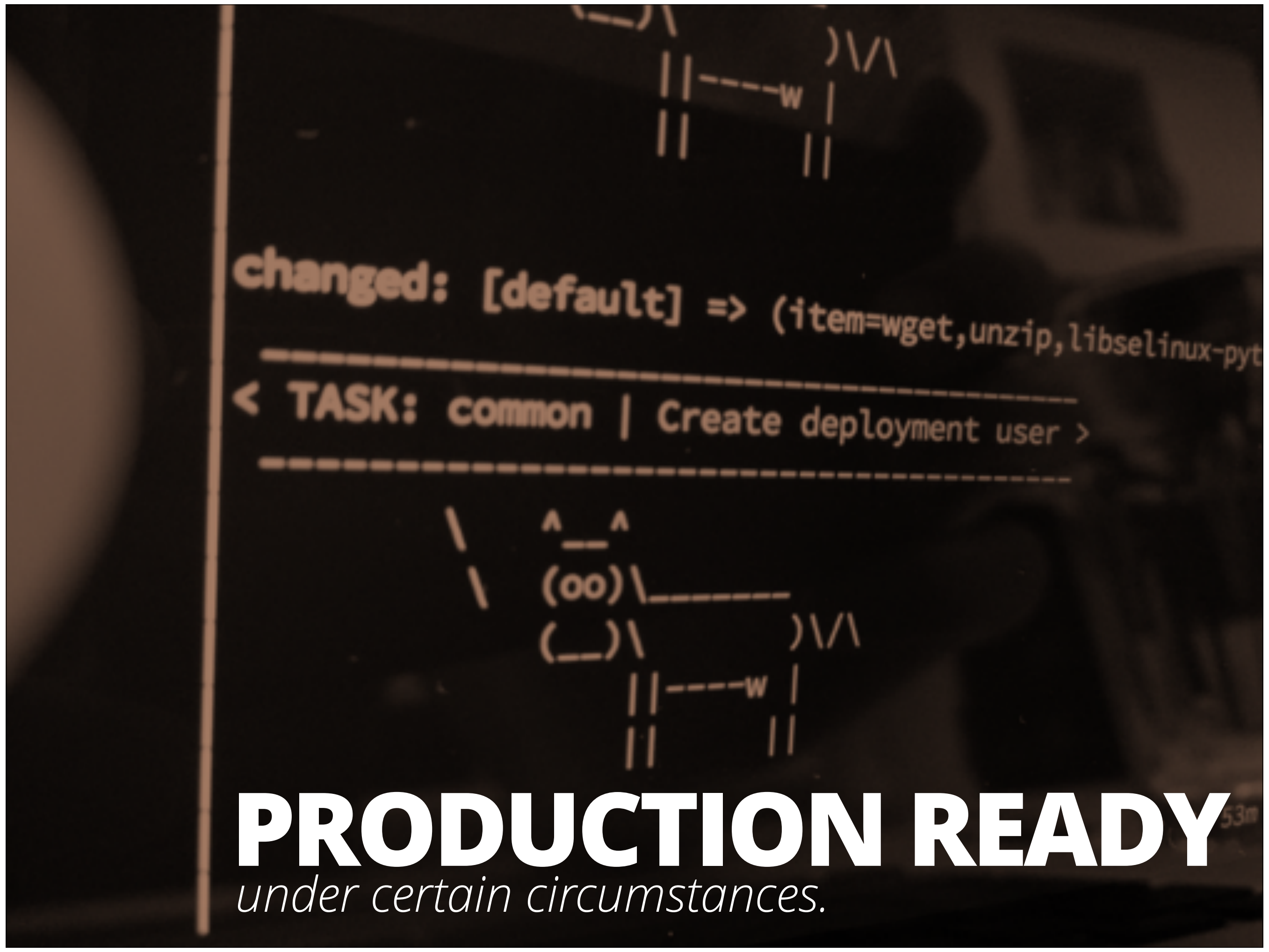
▸ Comments

Problem Details and System Configuration

```
Process:         Xcode [10455]
Path:            /Applications/Xcode.app/Contents/MacOS/Xcode
Identifier:      com.apple.dt.Xcode
Version:         6.0.1 (6528)
Build Info:      IDEFrameworks-6528000000000000~2
App Item ID:     497799835
App External ID: 712682811
Code Type:       X86-64 (Native)
Parent Process:  launchd [285]
Responsible:     Xcode [10455]
User ID:         501

Date/Time:       2014-10-22 14:21:57.928 -0400
OS Version:      Mac OS X 10.9.5 (13F34)
Report Version:  11
Anonymous UUID:  428438C0-D6CA-6FF4-A353-AF44D88E1538

Sleep/Wake UUID: A5238A59-A87F-471C-AA42-CF1D900AD92A

Crashed Thread:  11  Dispatch queue: com.apple.root.default-priority

Exception Type:  EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000

Application Specific Information:
ProductBuildVersion: 6A317
UNCAUGHT EXCEPTION (NSInvalidArgumentException): *** -[__NSPlaceholderDictionary initWithObjects:forKeys:count:]: attempt to insert nil object
objects[1]
UserInfo: (null)
Hints: None
Backtrace:
  0  0x00007fff88003244 __exceptionPreprocess (in CoreFoundation)
  1  0x0000000107b44184 DVTFailureHintExceptionPreprocessor (in DVTFoundation)
  2  0x00007fff88d75e75 objc_exception_throw (in libobjc.A.dylib)
  3  0x00007fff87f02dd1 -[__NSPlaceholderDictionary initWithObjects:forKeys:count:] (in CoreFoundation)
  4  0x00007fff87f18ad9 +[NSDictionary dictionaryWithObjects:forKeys:count:] (in CoreFoundation)
  5  0x0000000109005535 __85-[IDEDistributionSigningAssetsStepViewController _attemptToResolveProvisioningError:]_block_invoke_2 (in IDEKit)
  6  0x0000000107b7abac __DVTDispatchAsync_block_invoke (in DVTFoundation)
  7  0x00007fff8ef921bb _dispatch_call_block_and_release (in libdispatch.dylib)
  8  0x00007fff8ef8f28d _dispatch_client_callout (in libdispatch.dylib)
  9  0x00007fff8ef91082 _dispatch_root_queue_drain (in libdispatch.dylib)
 10  0x00007fff8ef92177 _dispatch_worker_thread2 (in libdispatch.dylib)
 11  0x00007fff90032ef8 _pthread_wqthread (in libsystem_pthread.dylib)
 12  0x00007fff90035fb9 start_wqthread (in libsystem_pthread.dylib)
```

My last project started in 2014 and decided to go Swift only. What this meant in the early months of Swift is a lesson we all learned the hard way. We saw this screen more often than we would have liked to.

And we also saw this little friend way longer than we’d love to.



```
changed: [default] => (item=wget,unzip,libselinux-pyt
 ______________________________________________
< TASK: common | Create deployment user >
 ----------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

PRODUCTION READY

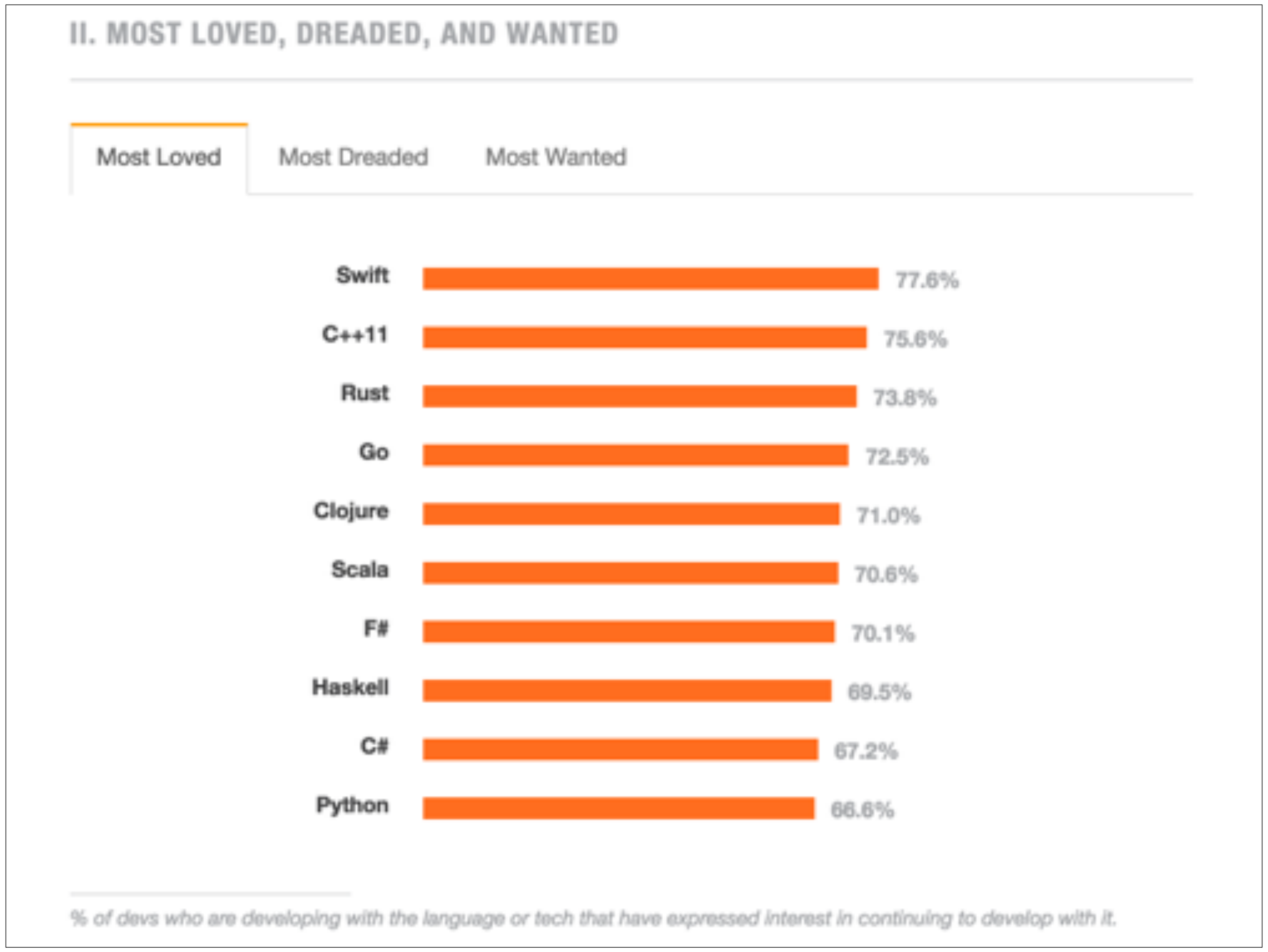
under certain circumstances.

In short words: Swift was there, but it was not quite what you would call production ready. It was still in Beta early in our project for what it's worth. But also the 1.0 version that arrived in September was not that much better. Long compile times, frequent Xcode crashes, missing autocompletion (SourceKit crashes, hello!) were giving us a really hard time.

On top of that there were frequent and big syntax changes with each Beta update of Xcode that proved to be quite a lot of work.

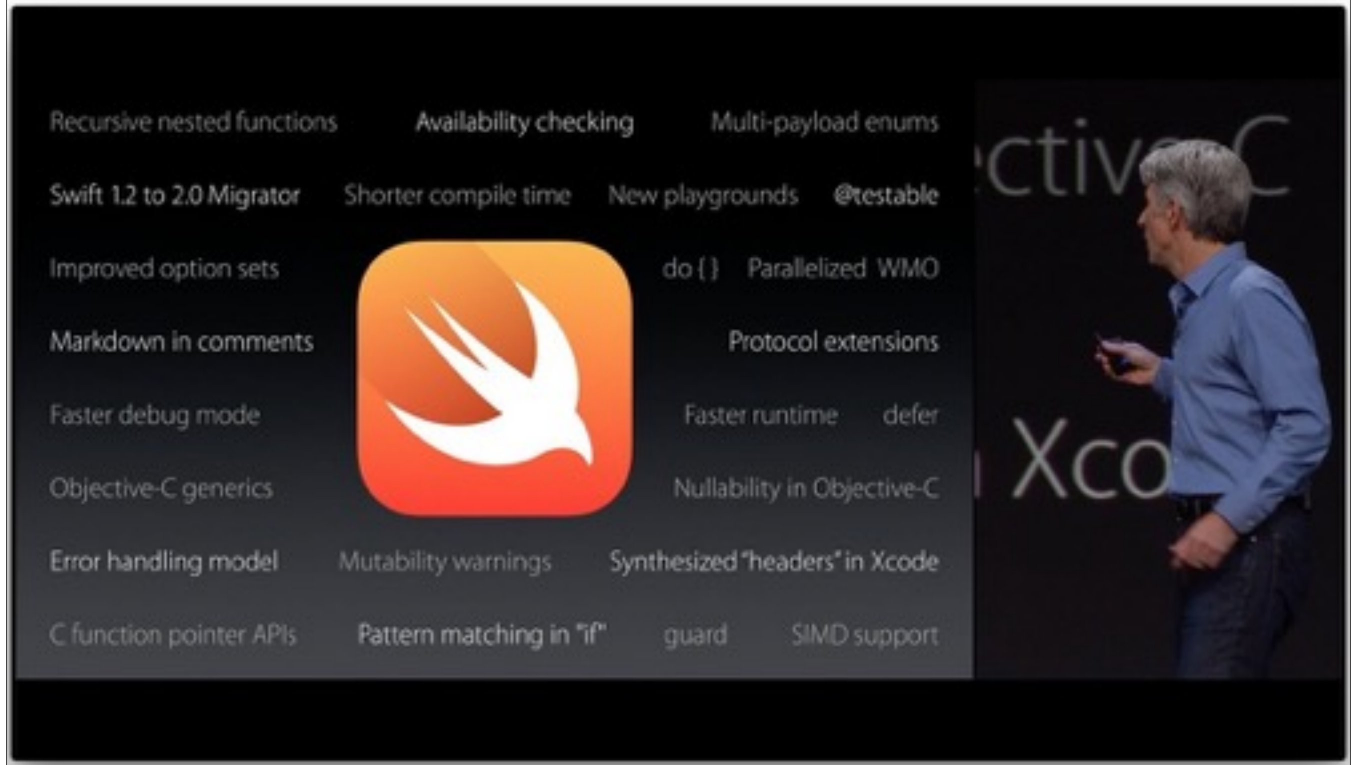
FAST FORWARD *TO 2015*

A lot has happened since 2014, let's see where we are standing today

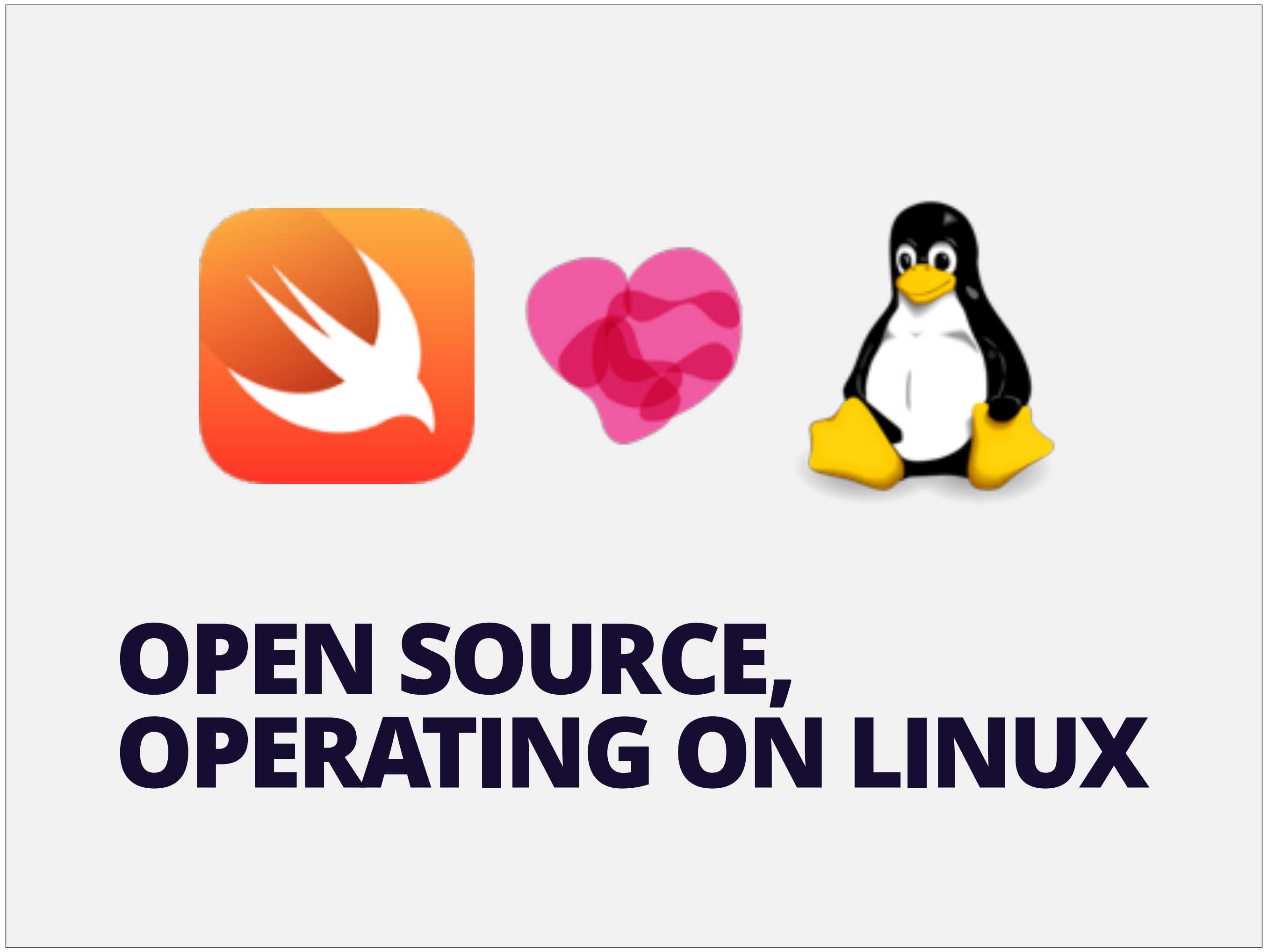


According to a survey on StackOverflow Swift seems to be the “Most Loved” language of 2014/2015 — whatever that means, it shows that people seem to be curious and excited to start using Swift.

At WWDC 2015 Apple announced Swift 2.0



It's supposed to be more stable, more mature and to bring a couple of improvements and new features. We'll take a deeper look at what the most exciting improvements and new features are in a minute.



OPEN SOURCE, OPERATING ON LINUX

One of the most exciting announcements to me as a Linux enthusiast and Open Source fanatic was Apple’s announcement to make Swift Open Source by the end of 2015 and to make the Swift compiler to play nicely with Linux. I’m still curious if Apple will keep their promise.

So…

WHAT'S NEW IN SWIFT 2.0 AFTER ALL?

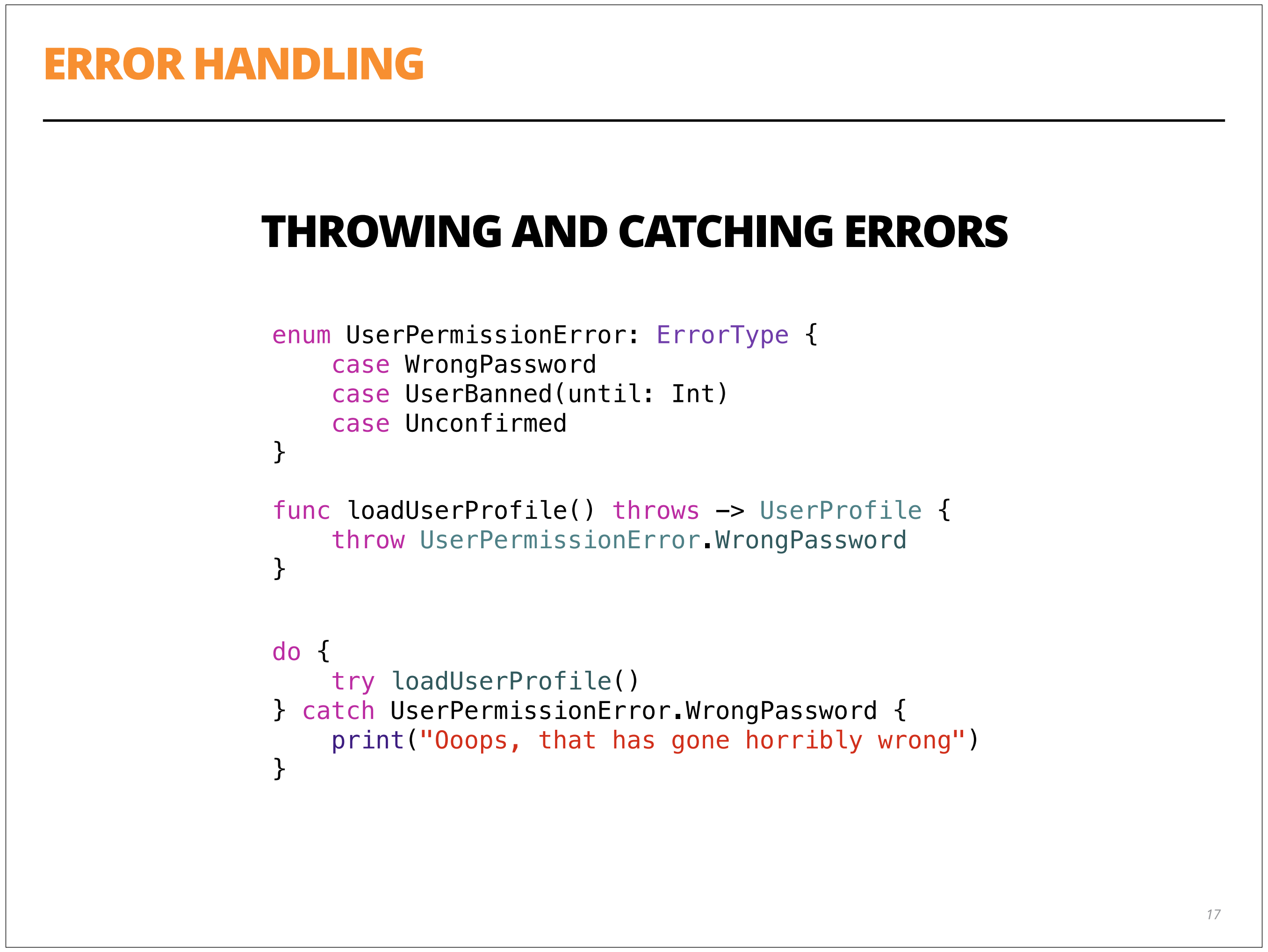
So, enough about the chit chat about Swift's history. We're here to learn about Swift 2.0

The upcoming slides are full of small and easy digestible code samples. I want to make this quite interactive so whenever there are questions just feel free to interrupt me and I'll try to explain a little more. I've also prepared a Playground that allows us to play interactively with the code samples given on the slides. Let's hope this works!

ERROR HANDLING & CONTROL FLOW

The first topic I want to talk about is the new Error Handling mechanisms as well as some new means for control flow.

Error *Handling*

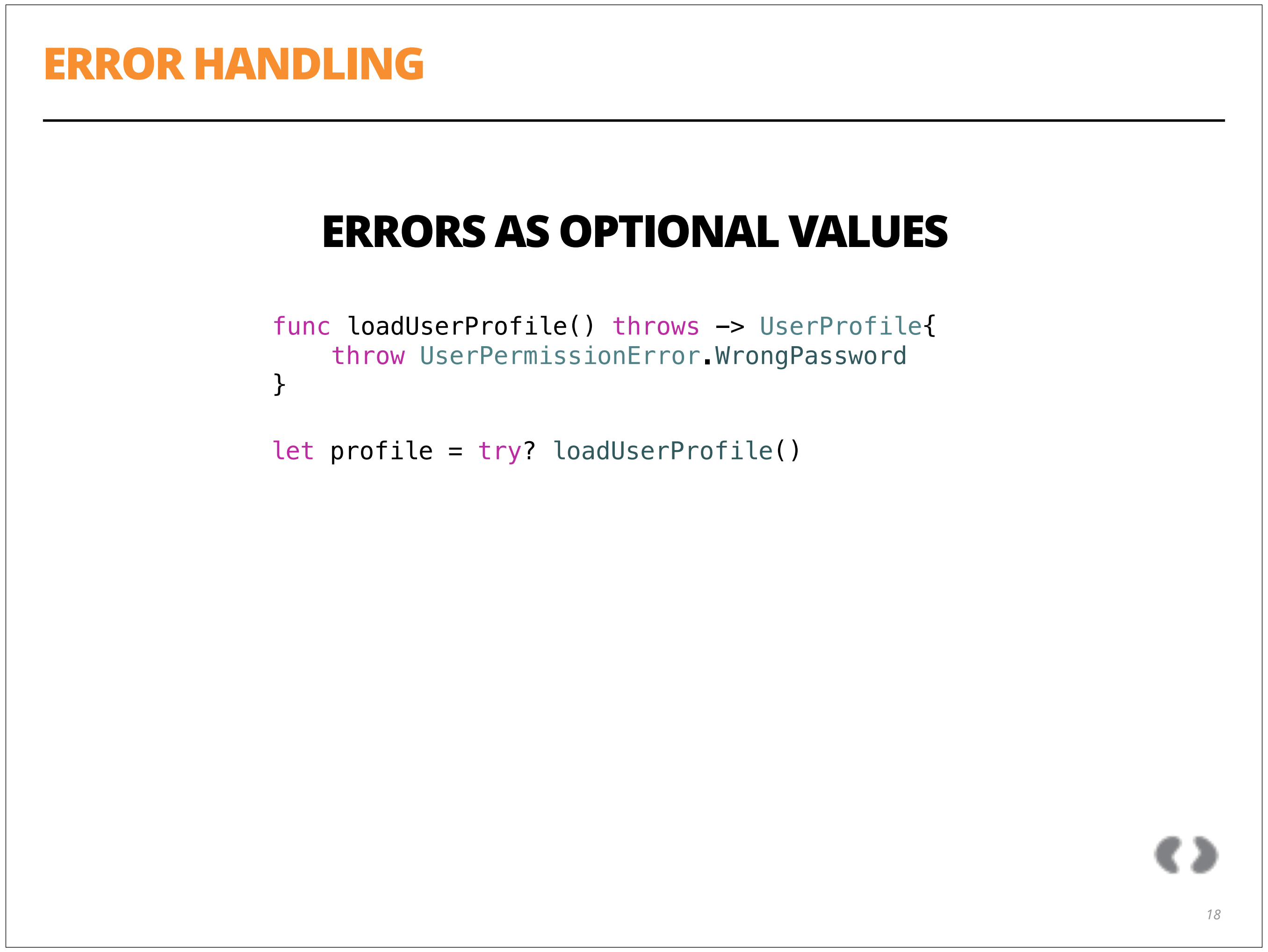


ERROR HANDLING

THROWING AND CATCHING ERRORS

```
enum UserPermissionError: ErrorType {
    case WrongPassword
    case UserBanned(until: Int)
    case Unconfirmed
}

func loadUserProfile() throws -> UserProfile {
    throw UserPermissionError.WrongPassword
}


do {
    try loadUserProfile()
} catch UserPermissionError.WrongPassword {
    print("Ooops, that has gone horribly wrong")
}
```

Similar to other languages you might know Swift has finally introduced ErrorTypes that come pretty close to what you might know from Java or C# in the form of Exceptions. You can throw them to interrupt your program flow, you can catch them to handle them appropriately. Objective-C has had this in the form of NSErrors but Swift was lacking. This forced Swift developers to either build their own error return types or to tread errors by registering error callbacks (or Promises/Futures that follow the same pattern)



ERROR HANDLING

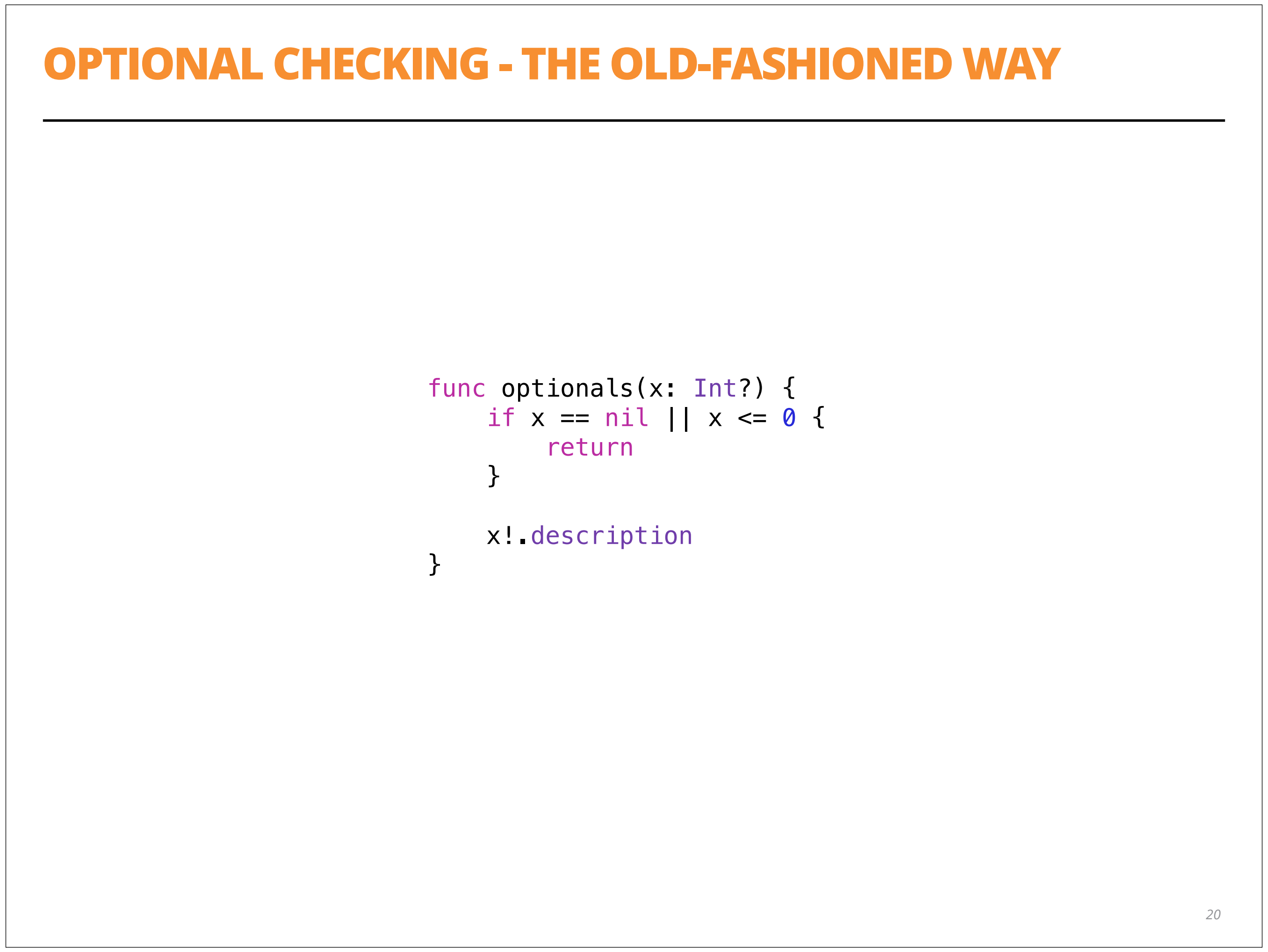
ERRORS AS OPTIONAL VALUES

```
func loadUserProfile() throws -> UserProfile{
    throw UserPermissionError.WrongPassword
}

let profile = try? loadUserProfile()
```

If you just don’t want to care about the types of errors you can treat exceptions by mapping them to an optional type. This could come in handy if you want to introduce ErrorTypes to an existing codebase without bothering about explicit error handling in the first step.

*Optional Checking and the **Guard** Statement*

There’s a new kid on the block for checking optionals and ensuring the condition of variables within your functions. The Guard statement.



OPTIONAL CHECKING - THE OLD-FASHIONED WAY

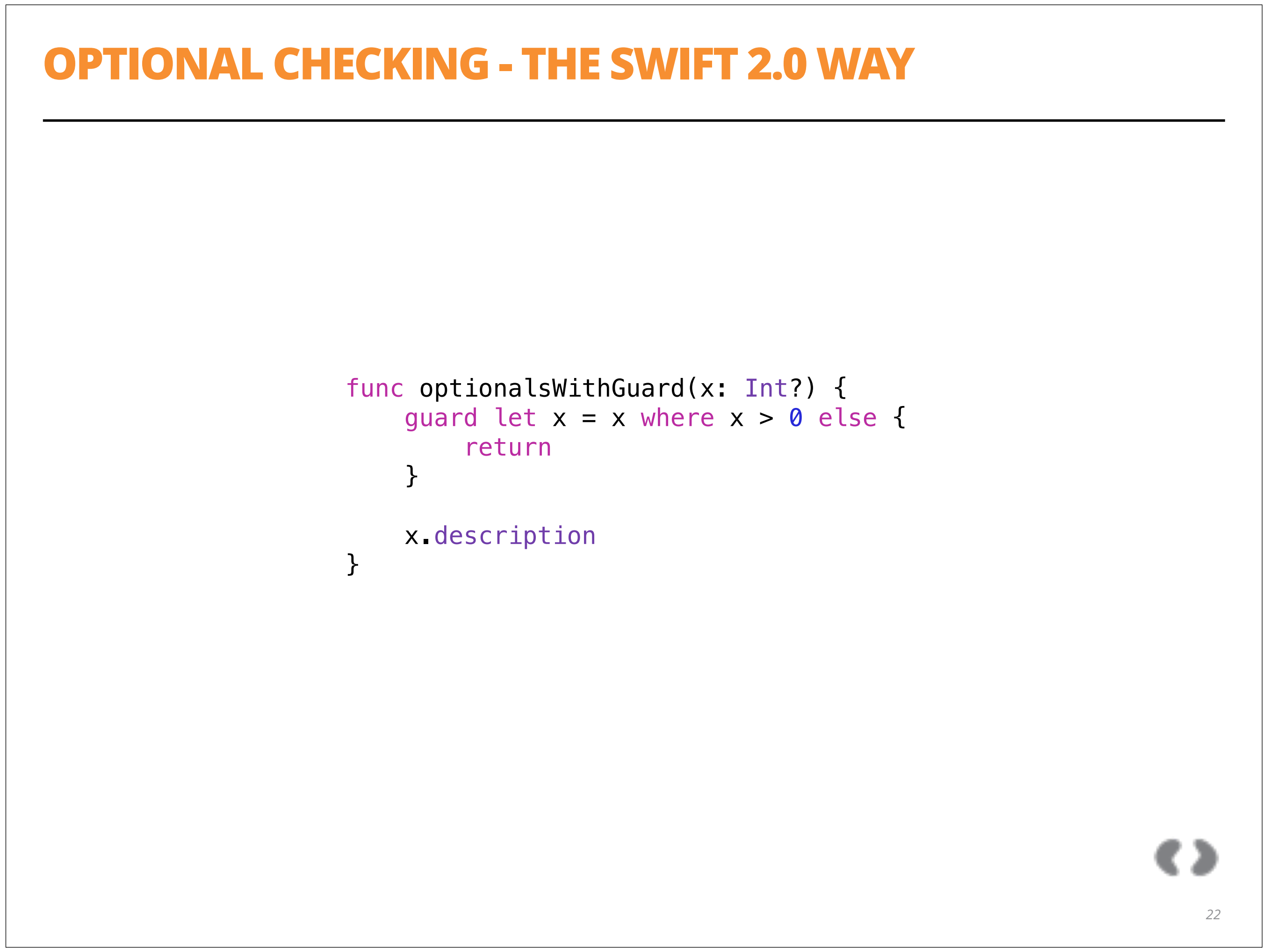
```swift
func optionals(x: Int?) {
    if x == nil || x <= 0 {
        return
    }

    x!.description
}
```

The old (Objective-C) way of checking for optionals looks rather like this.

OPTIONAL CHECKING - THE SWIFT 1.2 WAY

```swift
func optionalsWithIfLet(x: Int?) {
    if let myX = x where myX > 0 {
        myX.description
    }
}
```

With Swift you got the if let construct which was later even extended to be used with where clauses



OPTIONAL CHECKING - THE SWIFT 2.0 WAY

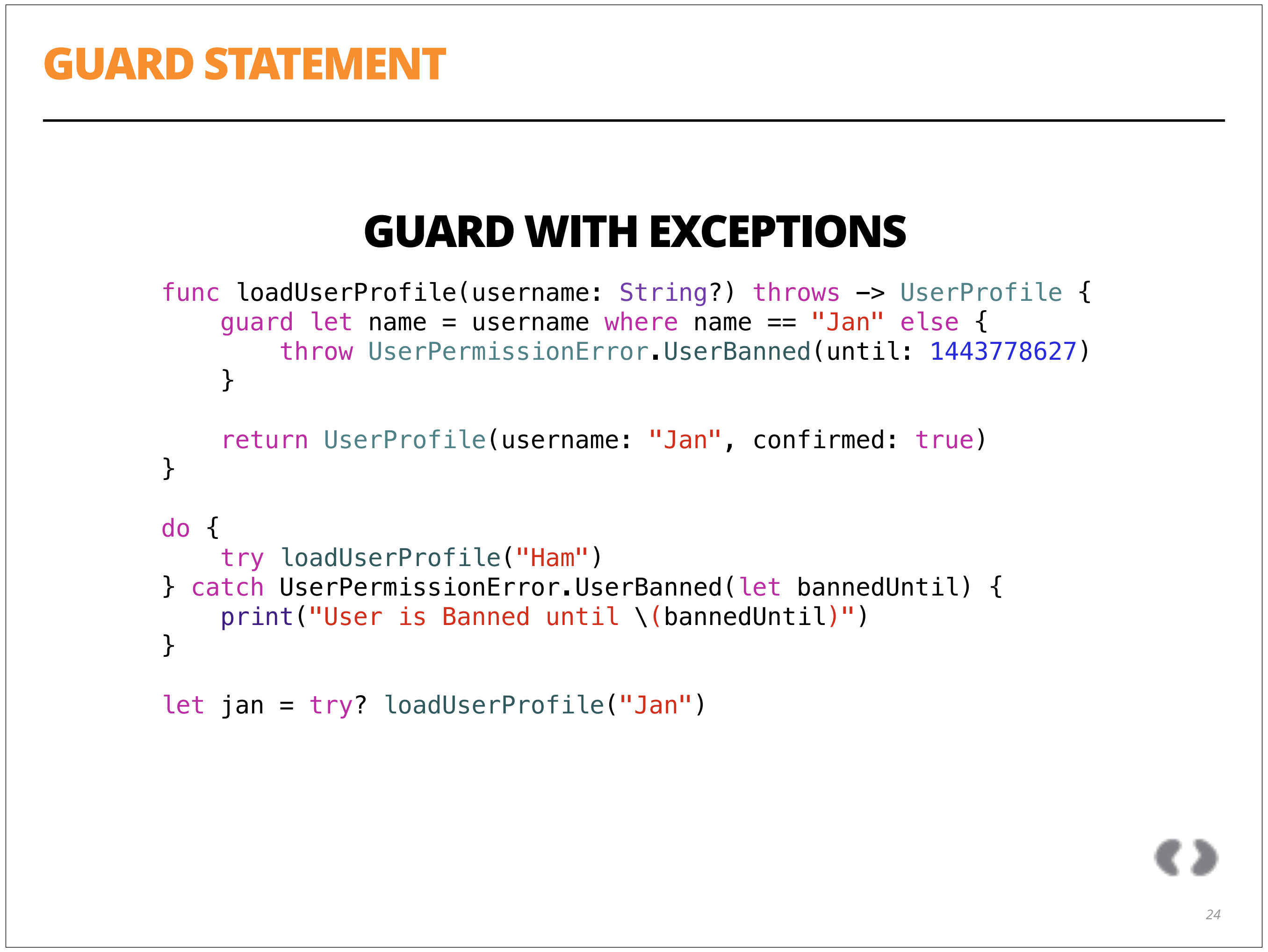
```swift
func optionalsWithGuard(x: Int?) {
    guard let x = x where x > 0 else {
        return
    }

    x.description
}
```

Guards will automatically unwrap the optional variables and allows you to handle the positive case outside of your guard clause

GUARD STATEMENT

SIMPLE GUARD STATEMENT

```
func loadUserProfile(username: String?) -> UserProfile? {
    guard let name = username where name == "Jan" else {
        return nil
    }

    return UserProfile(username: "Jan", confirmed: true)
}

loadUserProfile("Ham") // nil
loadUserProfile("Jan") // UserProfile(username: Jan, confirmed: true)
```

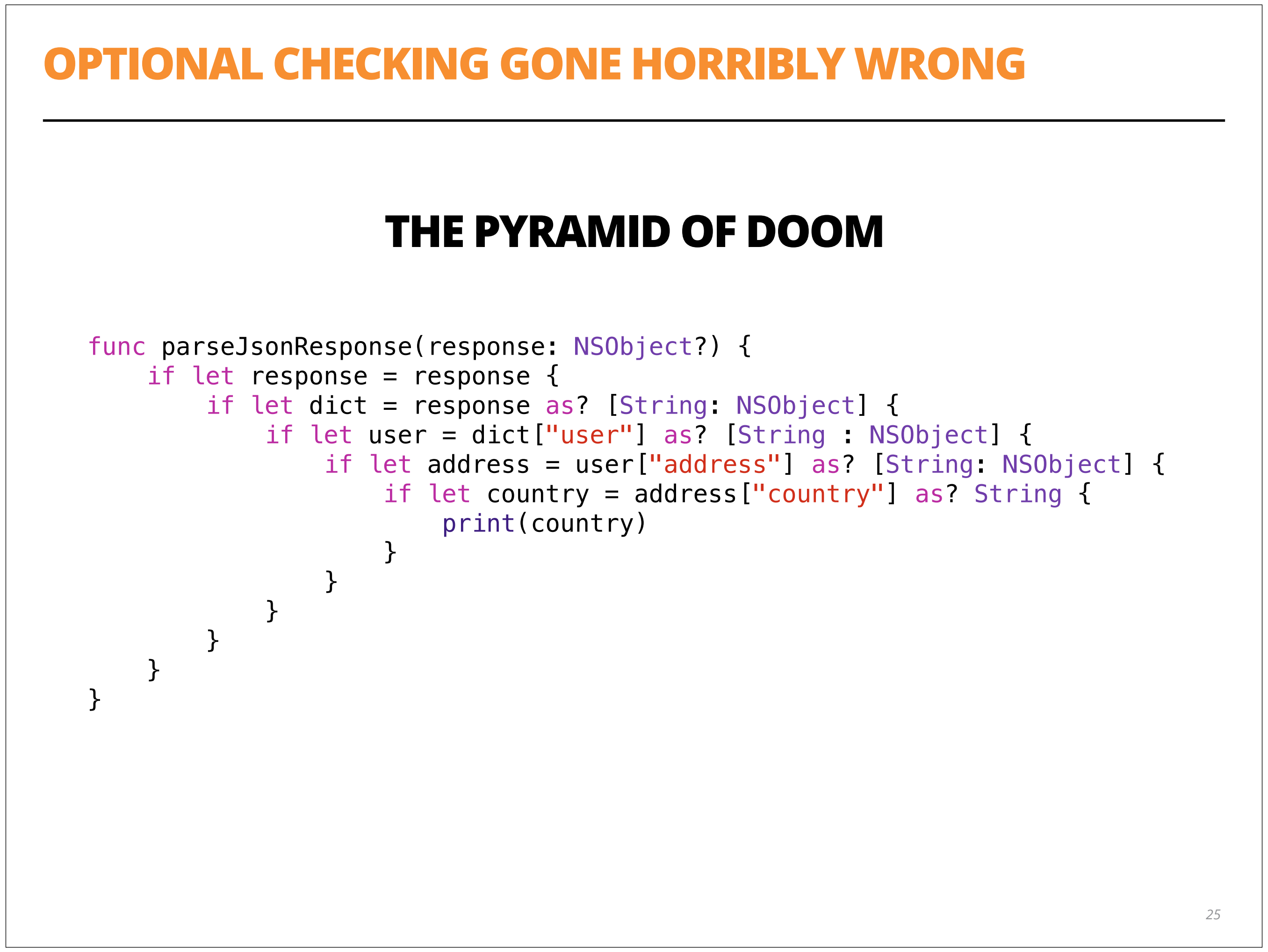


GUARD STATEMENT

GUARD WITH EXCEPTIONS

```
func loadUserProfile(username: String?) throws -> UserProfile {
    guard let name = username where name == "Jan" else {
        throw UserPermissionError.UserBanned(until: 1443778627)
    }

    return UserProfile(username: "Jan", confirmed: true)
}

do {
    try loadUserProfile("Ham")
} catch UserPermissionError.UserBanned(let bannedUntil) {
    print("User is Banned until \(bannedUntil)")
}

let jan = try? loadUserProfile("Jan")
```

Guards can be used with every kind of control flow statement, so ErrorTypes are naturally a very good fit to exit the function if something unusual occurs



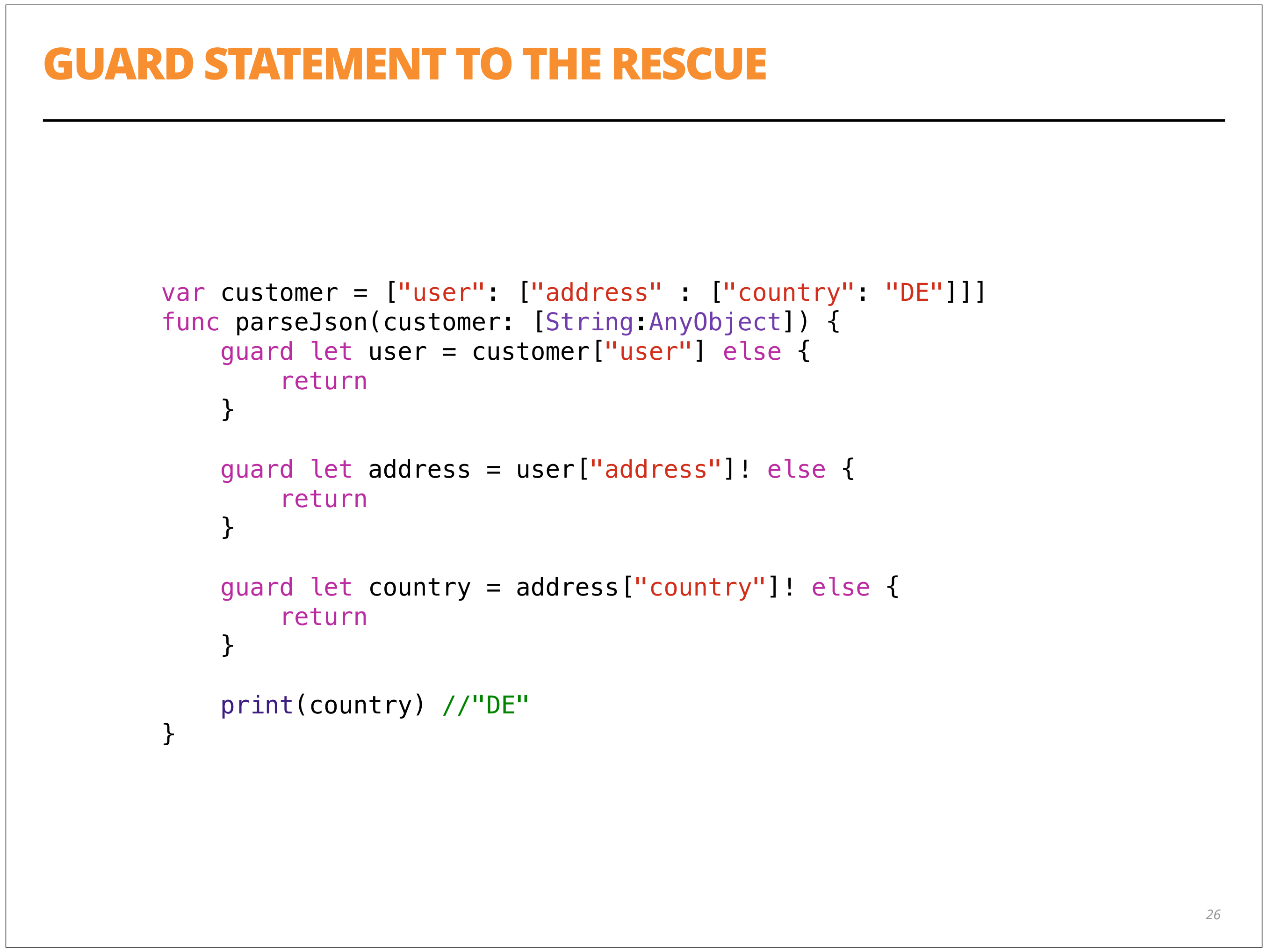
OPTIONAL CHECKING GONE HORRIBLY WRONG

THE PYRAMID OF DOOM

```swift
func parseJsonResponse(response: NSObject?) {
    if let response = response {
        if let dict = response as? [String: NSObject] {
            if let user = dict["user"] as? [String : NSObject] {
                if let address = user["address"] as? [String: NSObject] {
                    if let country = address["country"] as? String {
                        print(country)
                    }
                }
            }
        }
    }
}
```

This is a real-world example of where Optional Checking has gone horribly wrong (and its not even the complete example).

Since if let-statements do not expose the unwrapped optionals to the outside scope you had to work with those unwrapped and bound optionals inside if your if let block which often resulted in something as nasty as this. With Swift 1.2 we got the chance to do multiple bindings within one if let statement which at least helped avoiding that deep nesting



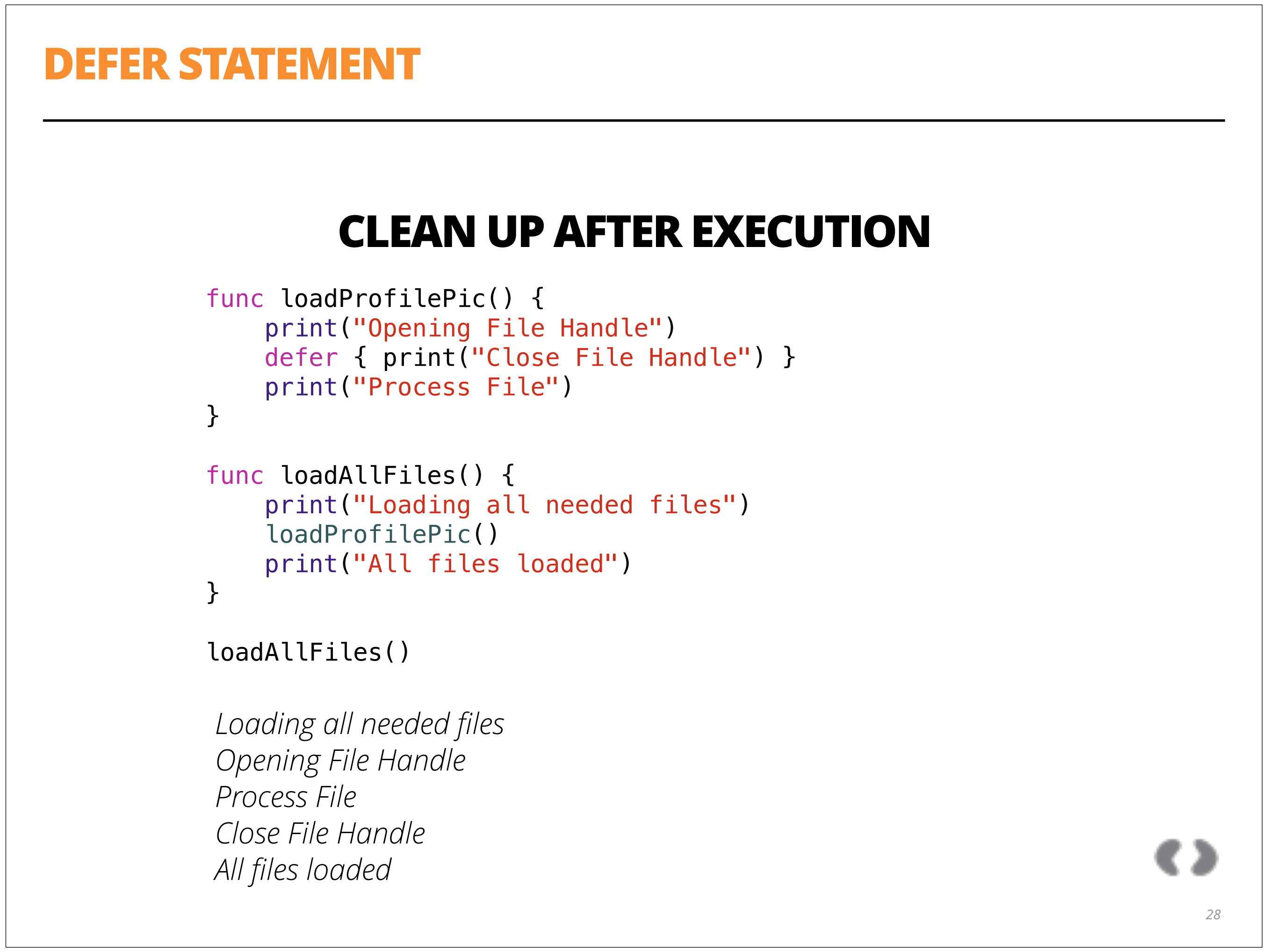
GUARD STATEMENT TO THE RESCUE

```
var customer = ["user": ["address" : ["country": "DE"]]]
func parseJson(customer: [String:AnyObject]) {
    guard let user = customer["user"] else {
        return
    }

    guard let address = user["address"]! else {
        return
    }

    guard let country = address["country"]! else {
        return
    }

    print(country) //"DE"
}
```

Guard statements expose the unwrapped and bound variable to the outside scope so that you can have quite nice and flat statements

Defer *Statement*

Swift 2.0 also introduced the Defer statement, which is comparable to the finally statement you might know from other languages



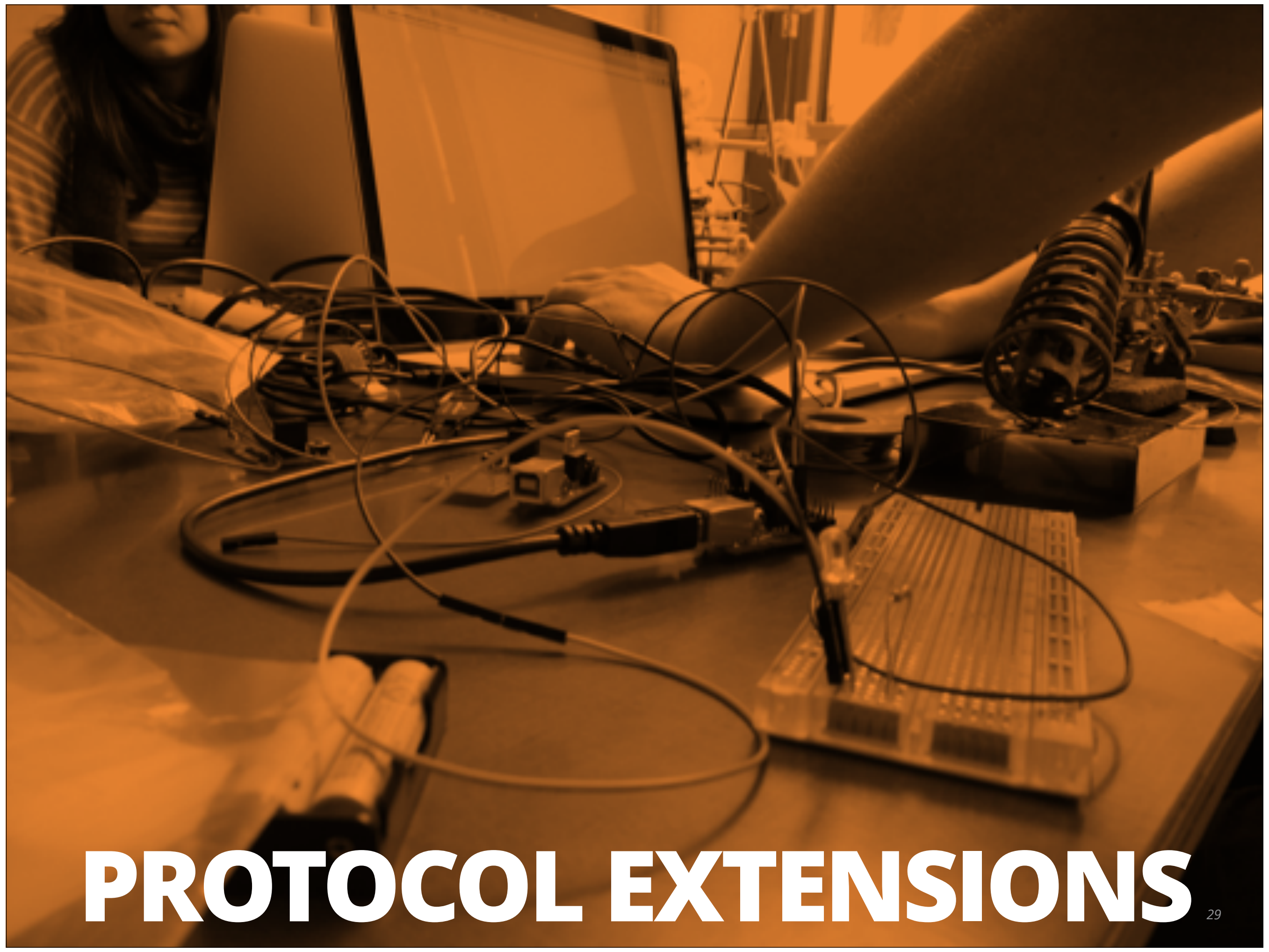
DEFER STATEMENT

CLEAN UP AFTER EXECUTION

```
func loadProfilePic() {
    print("Opening File Handle")
    defer { print("Close File Handle") }
    print("Process File")
}

func loadAllFiles() {
    print("Loading all needed files")
    loadProfilePic()
    print("All files loaded")
}

loadAllFiles()
```

```
Loading all needed files
Opening File Handle
Process File
Close File Handle
All files loaded
```

Defer is Swift’s “finally” statement and will be executed whenever the scope it’s defined in is about to be left.

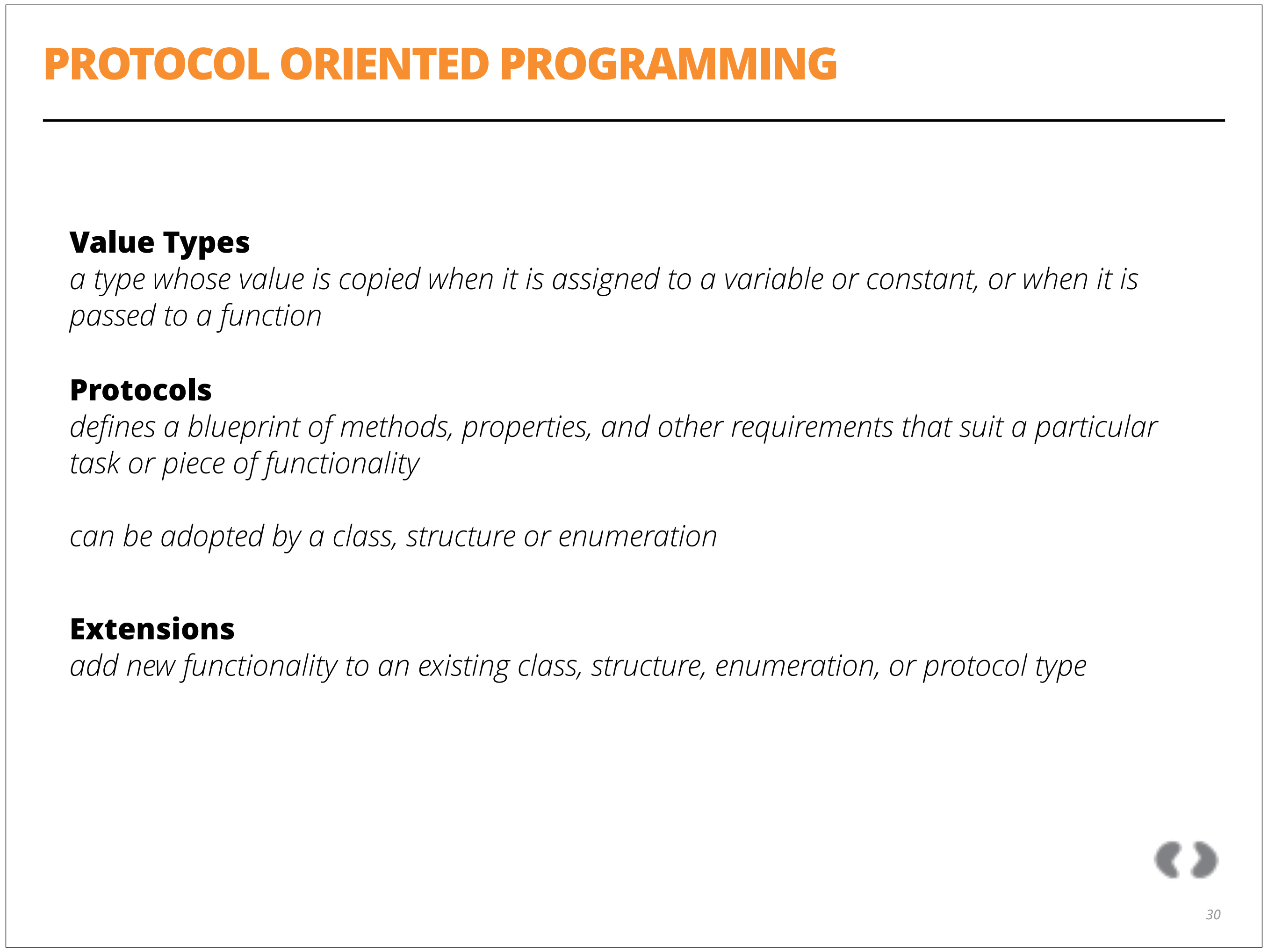


PROTOCOL EXTENSIONS

Another big novelty in Swift 2.0 is called Protocol Extensions. Sorry, I couldn't think of a better picture for this. So I just took a photograph from our last hardware hacking workshop to depict something that looks as complicated as the phrase Protocol Extensions sounds.

Protocol Extensions are the foundation for something that Apple calls "Protocol Oriented Programming". In a typical Apple fashion they claim that this is the latest and hottest stuff around that — once again — changes everything. For everyone else who has been around a little longer and who has discovered the world of programming outside of Apple's walled garden this concept is rather familiar and takes ideas from Functional Programming, Immutable Datatypes, etc.

Protocol Extensions themselves are rather comparable with Scala's traits or Mixins in Ruby. The idea is not as new as Apple might want to sell it and certainly nothing groundbreaking. But it's the foundation for some quite fancy stuff and — as Apple announced with the release of Swift 2.0 — this will be the way to develop your iOS and Mac OS apps in the future



PROTOCOL ORIENTED PROGRAMMING

Value Types
a type whose value is copied when it is assigned to a variable or constant, or when it is passed to a function

Protocols
defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality

can be adopted by a class, structure or enumeration

Extensions
add new functionality to an existing class, structure, enumeration, or protocol type

My colleague Thilo gave a really really good talk about Protocol Oriented Programming in Swift two weeks ago at our Hamburg meetup. This talk alone was enough material to fill and entire evening so please understand that I cannot go that much into detail tonight. I'll cover some of the basics (blatantly stolen from Thilo's talk). If you're interested in more information on the topic I'll give you some further resources afterwards.

MORE INFO

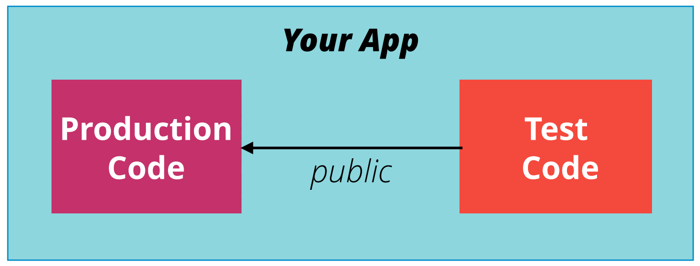
Protocol Extensions: https://developer.apple.com/videos/wwdc/2015/?id=408

Thilo's Talk on *Protocol Oriented Programming:*
https://speakerdeck.com/thilo/protocol-oriented-programming-in-swift
(make sure to checkout the interactive playground)

TESTING

With Swift 2.0 there have also been some changes around the way you can test your apps. Most notable are the @testable annotation and the new kind of UI Tests

SEPARATE TARGETS



PRODUCTION TARGET VS. TEST TARGET

Your typical app consists of your production code and your test code. The production code is the kind of stuff that makes your app, the stuff that you want to ship to your customer by putting it into the app store. The test code is for — well, for testing your production code.

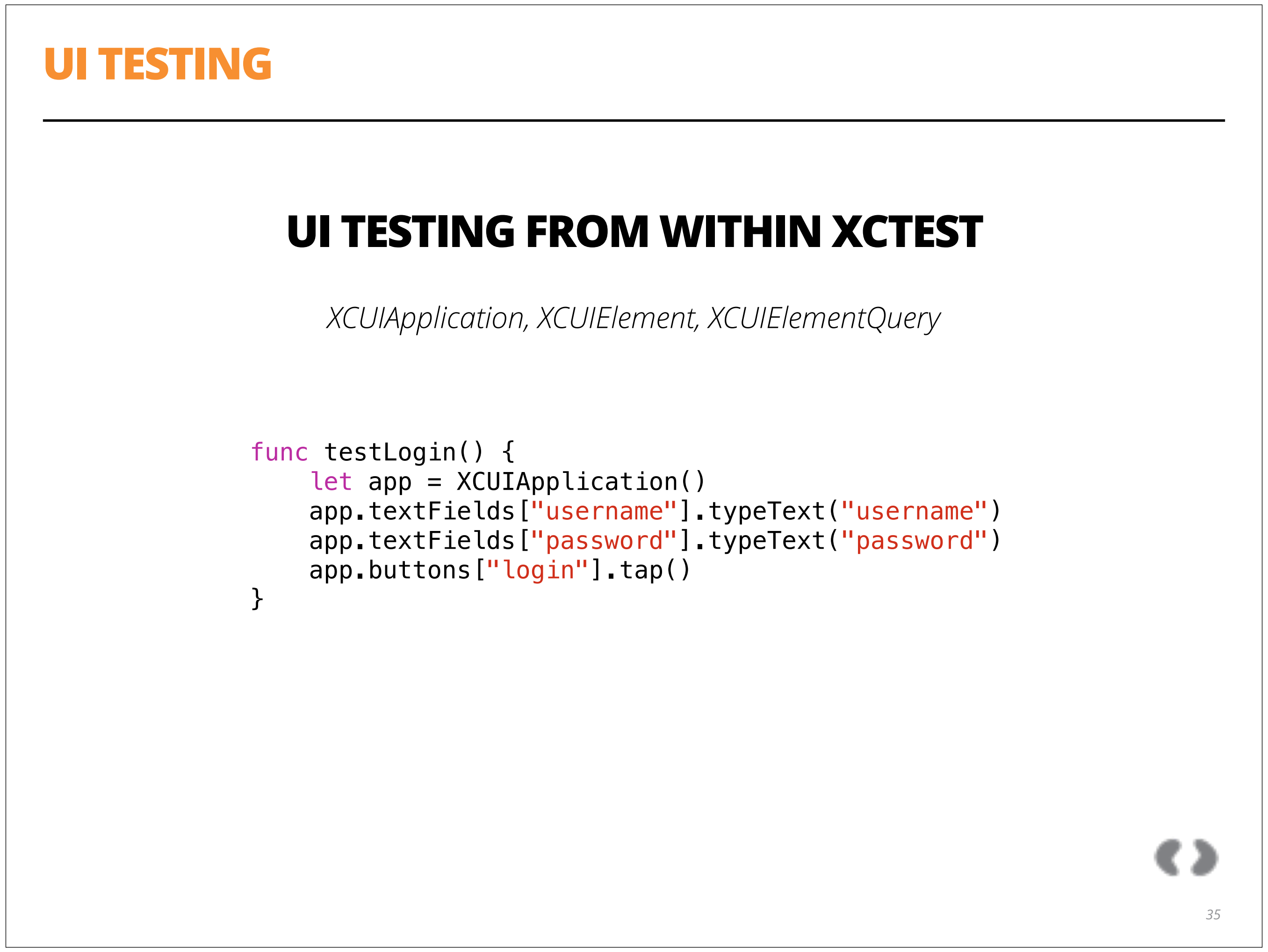
To reduce compile time you'll want to cleanly separate both as distinct targets within your app. This way you can compile your production target without needing to compile the test code.

If you have such a distinct separation, you can import your production code into your tests to access it from within your tests. However, this only allows you to access the public interface of your production code. You might want to test a little more than that without modifying the visibility of your production code. This is where @testable comes in.

@TESTABLE TO THE RESCUE

```
@testable import MyApp

func testMyApp() {
    // access internal properties, methods, and classes of MyApp
}
```

If you add @testable to your import, it allows you to also access internal properties, methods and classes of your production code



UI TESTING

UI TESTING FROM WITHIN XCTEST

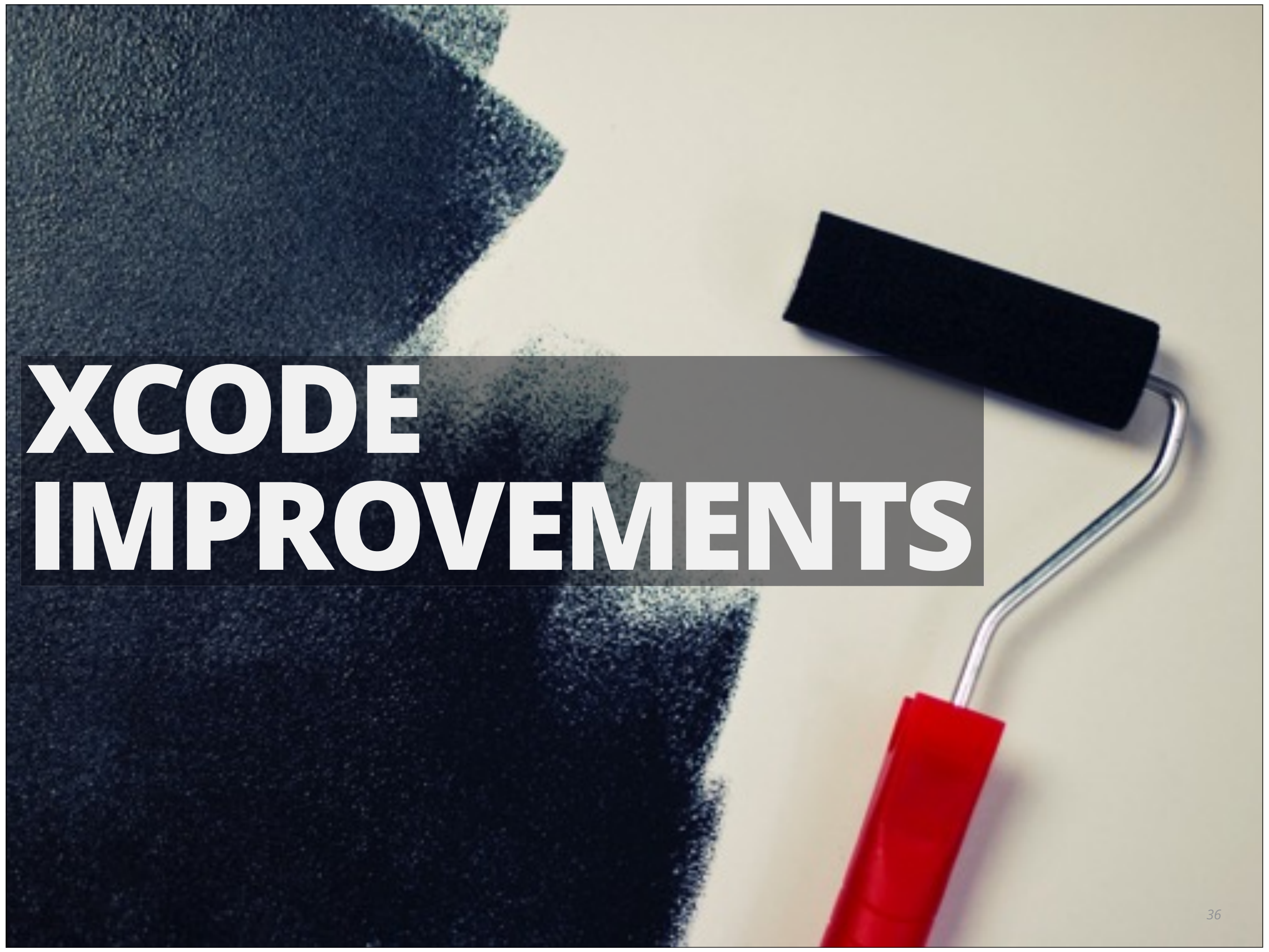
XCUIApplication, XCUIElement, XCUIElementQuery

```
func testLogin() {
    let app = XCUIApplication()
    app.textFields["username"].typeText("username")
    app.textFields["password"].typeText("password")
    app.buttons["login"].tap()
}
```

UI Testing is an important feature to test apps on a very high level. It allows testing user flows and UI Interactions the way a user would interact with the app. Usually you would use something as UIAutomation (which allows you to record tests interactively using Instruments) or one of the many frameworks out there.

With Swift 2.0 Apple extended XCTest to include some new classes to write UI Tests with XCTest. Note: This only works from iOS 9 onwards. In the latest stable version Xcode offers a record button similar to UIAutomation Instruments which works quite well.

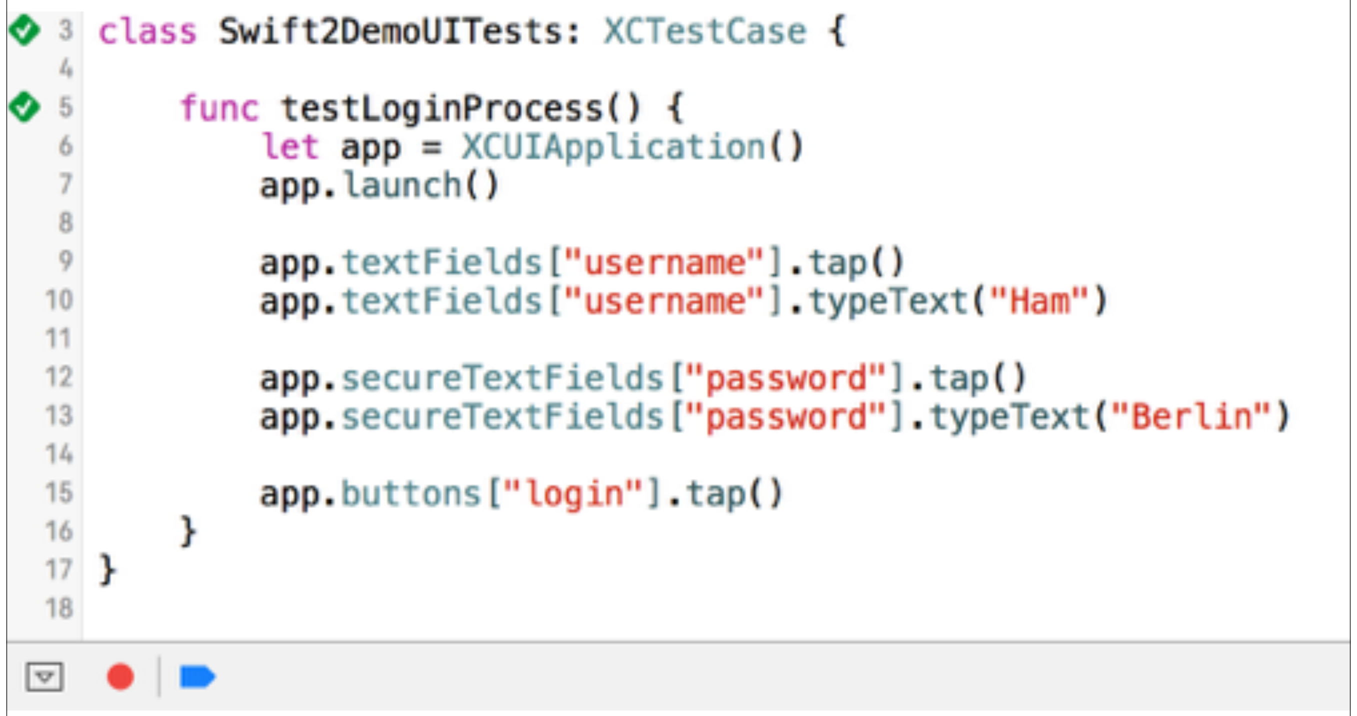
The information you can get from the elements found within the tests is rather limited. You cannot access all properties which prevents you from checking the actual state of your app. Most of these things can also be covered with conventional XCTests. UI Testing right now is rather suitable to check if you can actually complete user flows within your app.



XCODE IMPROVEMENTS

Since Swift and Xcode are tightly bundled an update to Swift always also means an update to Xcode. Along with the Xcode update to version 7 there have also been some improvements and new features to the IDE. I'll only take a glance at the major ones.

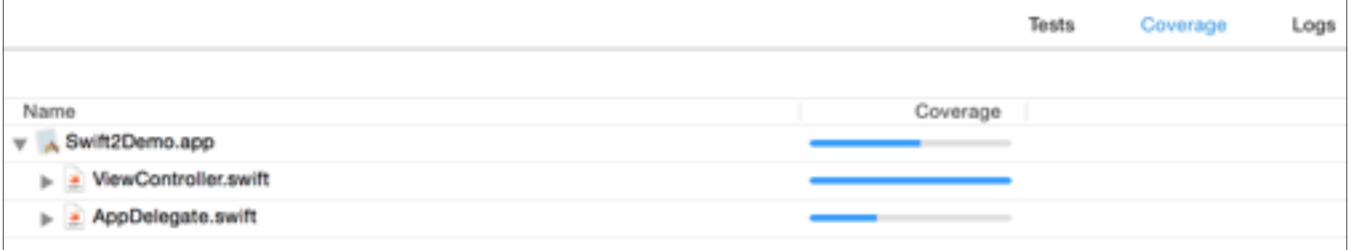
UI TEST RECORDING



```
class Swift2DemoUITests: XCTestCase {

    func testLoginProcess() {
        let app = XCUIApplication()
        app.launch()

        app.textFields["username"].tap()
        app.textFields["username"].typeText("Ham")

        app.secureTextFields["password"].tap()
        app.secureTextFields["password"].typeText("Berlin")

        app.buttons["login"].tap()
    }
}
```

Along with the new UI Tests, Xcode also allows you to record those test from within your IDE. This is quite similar to what UI Automation Instruments does as well but is actually a lot simpler. Simply click on the red “record” button, your app will be launched and you can tap around in your Simulator. Xcode will automatically record your taps and translate them into UI Test code.

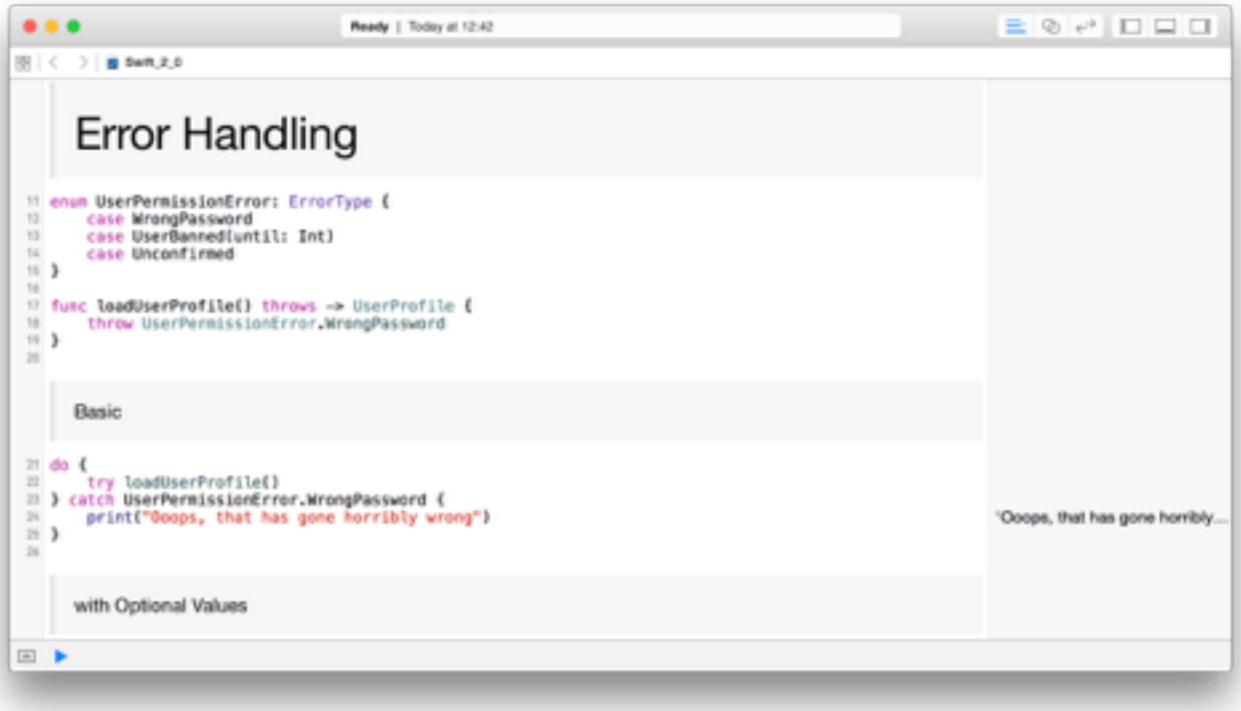
TEST COVERAGE REPORTS



Name	Coverage
Swift2Demo.app	
ViewController.swift	
AppDelegate.swift	

Xcode also offers Code Coverage reports to show how good your test suite covers your actual code. Simply activate Code Coverage reports in your Scheme, run your test and go to the details of your last test run.

PLAYGROUND IMPROVEMENTS



Error Handling

```swift
enum UserPermissionError: ErrorType {
    case WrongPassword
    case UserBanned(until: Int)
    case Unconfirmed
}

func loadUserProfile() throws -> UserProfile {
    throw UserPermissionError.WrongPassword
}

```

Basic

```swift
do {
    try loadUserProfile()
} catch UserPermissionError.WrongPassword {
    print("Ooops, that has gone horribly wrong")
}

```

"Ooops, that has gone horribly...

with Optional Values

As you have seen from my frequent switching to the playground, the playground itself has also been enhanced. It now supports Markdown in comments and offers a really nice way to showcase new stuff. It’s a fully functional REPL as you might know from Lisp, Ruby, Python, node and many others, that can also help you during your day to day programming

GROWN UP ECOSYSTEM

The Swift Ecosystem has grown significantly over the past year. In the beginning there was basically nothing, no libraries, no frameworks, no resources. In the meantime a huge amount of libs have been ported from Objective C, lots of new libs have been created.

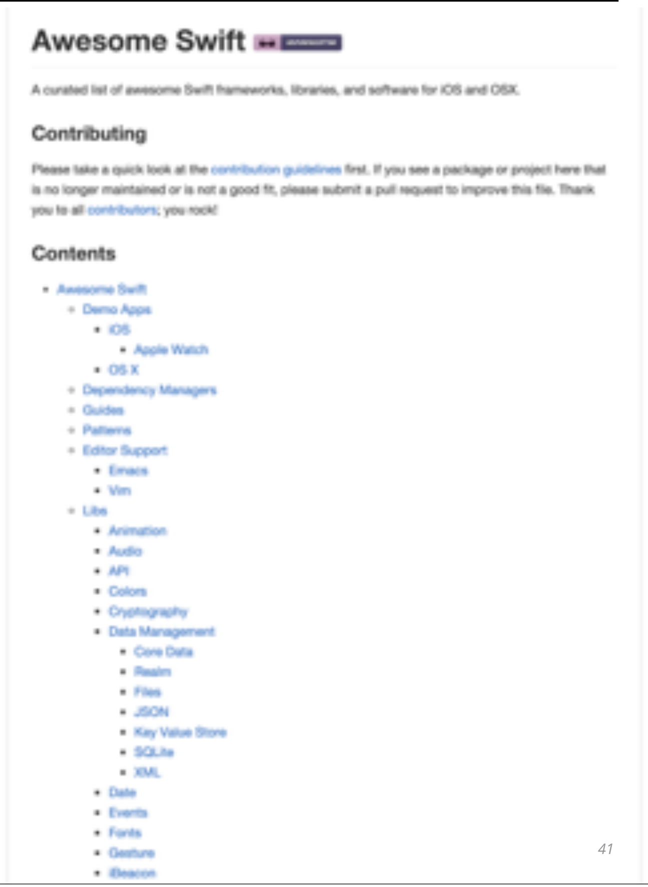
I used to tell people to think carefully if they want to adopt Swift for their projects. With the rich and great ecosystem we have at our fingertips now I no longer recommend anyone to be reluctant.

LIBRARIES, FRAMEWORKS, OTHER STUFF

AWESOME SWIFT

https://github.com/matteocrippa/awesome-swift

https://github.com/Wolg/awesome-swift



Awesome Swift

A curated list of awesome Swift frameworks, libraries, and software for iOS and OSX.

Contributing

Please take a quick look at the contribution guidelines first. If you see a package or project here that is no longer maintained or is not a good fit, please submit a pull request to improve this file. Thank you to all contributors; you rock!

Contents

- Awesome Swift
  - Demo Apps
    - iOS
      - Apple Watch
    - OS X
  - Dependency Managers
  - Guides
  - Patterns
  - Editor Support
    - Emacs
    - Vim
  - Libs
    - Animation
    - Audio
    - API
    - Colors
    - Cryptography
    - Data Management
      - Core Data
      - Realm
      - Files
      - JSON
      - Key Value Store
      - SQLite
      - XML
    - Date
    - Events
    - Fonts
    - Gesture
    - iBeacon

RECAP - WHAT'S NEW?

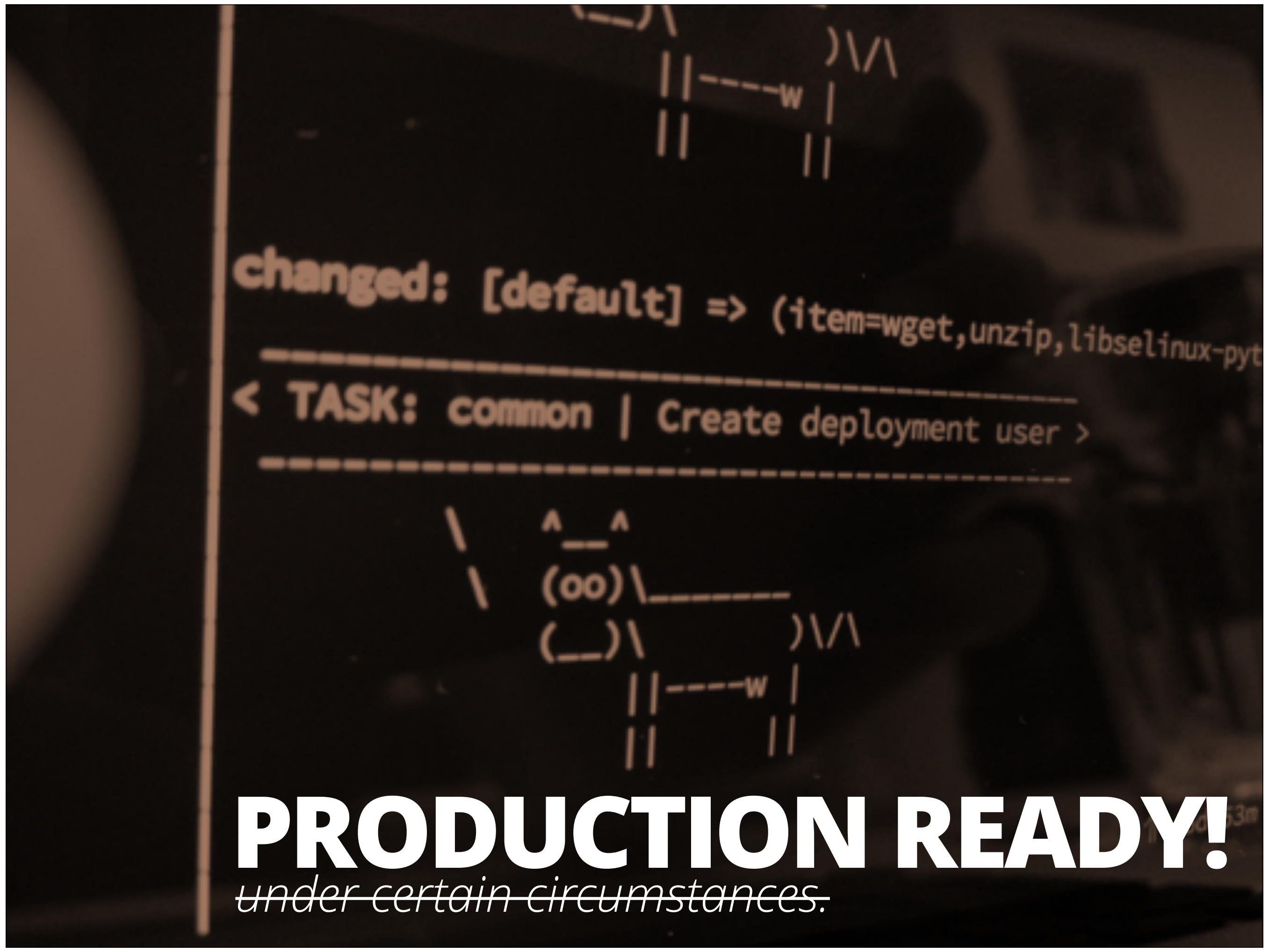
New ***Error Handling*** *mechanisms*

New ***Control Flow*** *Statements*

Protocol Extensions *- Focus on Protocol-Oriented Programming*

Testing - ***UITesting, @testable***

Xcode *Improvements*



```
    (__)\       )\/\
        ||----w |
        ||     ||

changed: [default] => (item=wget,unzip,libselinux-pyt
 ________________________________________________
< TASK: common | Create deployment user >
 ------------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

PRODUCTION READY!

~~under certain circumstances.~~

Swift has matured a lot over the past year. The IDE is pretty stable today, the syntax changes are only minor, Swift 2.0 apps can be submitted to the app store. If someone asks me today if Swift in general is a good idea to use exclusively for your new project I can go with a clear “yes”.

Sure, there are still some shortcomings, compile times are not amazing (at least compared with Java and other languages, but I’ve heard that it’s still better or at least comparable to Objective-C), there’s still no refactoring support in Xcode for Swift code. But there’s a rich ecosystem, a stable language with lots of good and well-thought features. So there’s that.

THANK YOU!

Questions, criticism and discussions welcome!

Follow me on Twitter: ***@hamvocke***