MODULE: Taking Your Apps to Production
SESSION [3]
SESSION TITLE: [Analyzing Your Application]

Sources:

1. [Professional IOS programming] [Chapter 15][9781118661130]
2. [iOS App Development][Chapter 12][9781118329894]

MODULE Objective

At the end of this module, you will be able to:

- Analyze your applications

Session Objectives

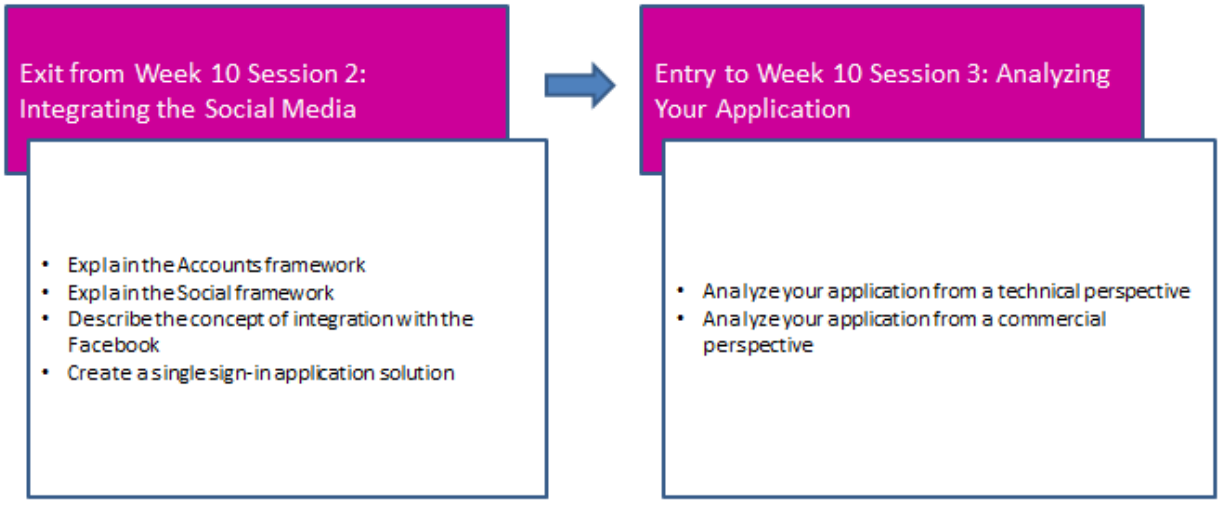
At the end of this session, you will be able to:

- Analyze your application from a technical perspective
- Analyze your application from a commercial perspective

<H1> Introduction

After all the hard work you've put into developing your iOS application, you want it to be a success. If you have developed the application for a paying customer, you probably are now in the phase where you can offer him/her a test version for evaluation purposes to see if your effort meets his/her expectations. However, if you have developed an application for yourself with the aim to monetize it, you also need to make sure the application is evaluated and tested before taking it into production.

This session shows you how to perform a technical analysis before taking an application into production. It also shows you how to implement a commercial analysis as a process to be executed after the application has been taken into production.



<H1> [Performing a Technical Analysis]

Source: [Professional iOS Programming] [Chapter 15] [Page 435]

When your application has finally been realized and is available in the App Store, you need to get users to download it and start using it on their device. Studies have revealed that when users download and install an application on their smartphones and launch it, they want it to operate immediately and smoothly. When an application crashes during the first-time usage, in most cases users will remove the application from their device. In addition, when the application is using extensive storage space or memory, the users will terminate and remove the application. All the money you might have spent in marketing your application to get a user to download and install it is lost. Even worse, you might receive a negative review.

The key technical reasons for applications to perform poorly or behave in a user-unfriendly manner are as follows:

- Application crashes
- Blocking the main thread
- Memory leaks
- Using synchronized HTTP requests
- Extensive bandwidth usage
- Battery drainage
- Unimpressive user interface

This section takes a closer look at these technical reasons so you can learn how to identify them and, where possible, avoid them.

<H2> [Application Crashes]

Source: [Professional iOS Programming] [Chapter 15] [Page 436]

Application crash is the number one reason for users to stop using an application. Crashes can happen for many different reasons, and most fall directly under the responsibility of the programmer.

To prevent application crashes, you can create a **crash handler**.

You can use a crash handler for your applications for the following two reasons:

- In case of an application crash, you can present a message to the user to inform him/her.
- You can collect information from the crash log and use a web service to send it to you for analysis.

For example, you can use **Redmine** (*www.redmine.org*) as the main management system. It is an open source platform to manage projects and tasks, and integrates with a ***subversion*** (also known as `svn`) system for source code management.

You can send the collected crash log directly to Redmine by calling a web service and creating a task of type crash in your system. If an application crashes on a user's device, you will be notified within a minute with the information of the device it has been running on, the version of the application, and the crash log. This will give you all the information you need to find a bug, fix it, and publish a new release of the application.

So far you have learned all the techniques and technologies to implement a similar process that fits in your organization.

<H2> [Blocking the Main Thread]

Source: [Professional iOS Programming] [Chapter 15] [Page 436]

A common mistake still made by many developers is blocking the main thread, which results in an unresponsive application. You should *never* perform heavy or long-running operations on the main thread of the application because `UIKit` does all of its work there. The easiest way to avoid this mistake is to move your operation to a background queue so it will not interfere with the main thread.

Create a `dispatch_queue_t` instance by calling the `dispatch_get_global_queue` function, which returns a global concurrent queue of a given priority level. Call the `dispatch_asynch` function to submit the block to the queue. A sample implementation is shown below:

```
dispatch_queue_t bgQueue = dispatch_get_global_queue
      (DISPATCH_QUEUE_PRIORITY_BACKGROUND,0);
dispatch_async(bgQueue, ^{
//perform your operation
});
```

Instead of calling the `dispatch_get_global_queue` function, you can also create your own queue by calling the `dispatch_queue_create` function with a name for the queue. A sample implementation is shown below:

```
dispatch_queue_t                              bgQueue                              =
dispatch_queue_create(“net.yourdeveloper.app.queuename”,nil);
dispatch_async(bgQueue, ^{
//perform your operation
});
```

<H2> [Memory Leaks]

Source: [Professional iOS Programming] [Chapter 15] [Page 437]

A memory leak occurs when an application incorrectly manages memory allocations. As a result, the object expected to be in memory cannot be accessed.

The main cause of memory leaks is when you manage the memory yourself in your application, your allocation and release of memory is not balanced. This might result in an application crash when you try to access an object that has already been released.

The introduction of **Automatic Reference Counting** (ARC) in iOS 5 has been a blessing for many programmers who were having a hard time in managing memory correctly in their applications. ARC implements automatic memory management for Objective-C objects and blocks, freeing the programmer from the need to explicitly insert retains and releases. It does not provide a cycle collector; users must explicitly manage the lifetime of their objects, breaking cycles manually or with weak or unsafe references. When two objects keep a reference to each other and are retained, it creates a retain cycle since both objects try to retain each other, making it impossible to release.

ARC may be explicitly enabled with the compiler `flag-fobjc-arc` and explicitly disabled with the compiler `flag -fno-objc-arc`.

It is important to realize that ARC is a feature of the **Objective-C compiler**, and therefore the entire ARC logic of releasing and retaining happens when you build your app. ARC is not a runtime feature, except for one small part: the weak pointer system. The only thing that ARC does is insert retains and releases into your code when it compiles it, exactly where you would have, or at least should have, put them yourself.

<H2> [Using Synchronized HTTP Requests]

Source: [Professional iOS Programming] [Chapter 15] [Page 438]

You already know how to make network calls to websites, **Restful services**, and **Simple Object Access Protocol** (SOAP) services.

There are two main objects involved in calling an HTTP service:

- **`NSURLRequest`**: The `NSURLRequest` object represents a URL load request independent of protocols and URL schemes. The main constructor accepts an `NSURL` object containing the URL to load, and optionally the cache policy for this request.
- **`NSURLConnection`**: The `NSURLConnection` object is used to perform loading of the `NSURLRequest`. The object can be invoked either synchronously or asynchronously, but the Apple developer guidelines strongly discourage the use of synchronous connections because your application thread will block until the response is received. This means that the complete user interface is blocked; if the connection cannot be established, your application thread only comes back alive after the timeout period has expired, which can be up to 900 seconds.

You can use the `URLRequest` class for your network communications to avoid the application from having to wait for the response of the request and becoming unresponsive.

<H2> [Extensive Bandwidth Usage]

Source: [Professional iOS Programming] [Chapter 15] [Page 438]

The immense success of smartphones and tablets and the enormous amount of free applications have resulted in mobile operators changing their terms for mobile contracts. Up until 2012, most mobile operators offered unlimited bandwidth with their prepaid and subscription plans. The explosive growth of the smartphone market consumed a lot of bandwidth of the operator networks, causing a negative effect on their earnings. As a result, most operators now offer only packages with a certain amount of megabytes (MB) in a certain timeframe, such as 500 MB/month. When users reach the agreed amount, they have to pay for each additional MB at a relatively high rate.

If your application uses a lot of bandwidth, resulting in users having to pay for additional MBs, they will not be happy and might even blame your application. Especially when your application is communicating with a web service, you have be **careful to avoid reloading the complete dataset each time the user launches your application**; you might consider **storing the received remote information using Core Data**.

Another important improvement is to **check if the user's device is connected via a Wi-Fi connection** or a cellular connection like 3G or 4G.

In most cases, a Wi-Fi connection is free—such as at home, work, or school, whereas the cellular connection is always related to the user's operator plan.

Apple has developed a sample application named **Reachability** that demonstrates how to use the `SystemConfiguration` framework to monitor the network state of a device. It contains a `Reachability.h` file and a `Reachability.m` file that you are free to use in your own applications. The `Reachability` class is not ARC-compliant, which means that if you use this class in one of your ARC projects, you must set the `-fno-objc-arc` compiler flag for the `Reachability.m` file in your project.

To demonstrate, start Xcode and create a new project using the **Single View Application** template. Name your project as **BatteryDrainer** using the options shown in **Figure 1**.

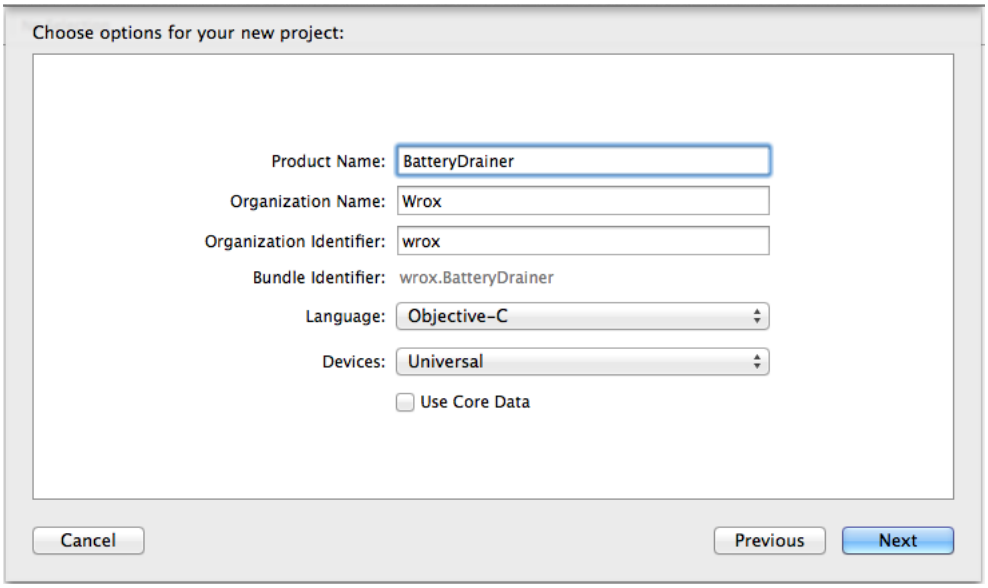


Figure 1: Create a New Project, BatteryDrainer, in Xcode

Download the Apple Reachability sample project to obtain a copy of the `Reachability.h` and `Reachability.m` files. Copy these two files to your project and add the `SystemConfiguration`

and CFNetwork frameworks. Set the -fno-objc-arc compiler flag for the Reachability.m file in your project.

Open the YDAppDelegate.h file and import the Reachability header file. Create a strong property of type Reachability named reachability and a public method named connectedViaWIFI, as shown below:

```
#import <UIKit/UIKit.h>

#import "Reachability.h"
@class YDViewController;

@interface YDAppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@property (strong, nonatomic) YDViewController *viewController;

@property (strong, nonatomic) Reachability* reachability;

-(BOOL)connectedViaWIFI;
@end
```

Open the YDAppDelegate.m file and create and initialize the reachability instance. Call the startNotifier method of the Reachability class. Implement the connectionViaWIFI method by returning the currentReachabilityStatus equal to the ReachableViaWIFI constant of your reachability instance.

The main part of the YDAppDelegate.m implementation is shown below:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]];
self.reachability = [Reachability reachabilityForInternetConnection];
[self.reachability startNotifier];
[self.reachability currentReachabilityStatus];

// Override point for customization after application launch.
self.viewController = [[YDViewController alloc]
```

```
      initWithNibName:@"YDViewController" bundle:nil];
self.window.rootViewController = self.viewController;
[self.window makeKeyAndVisible];
return YES;
}

-(BOOL)connectedViaWIFI
{
      return          [self.reachability          currentReachabilityStatus]==
      ReachableViaWiFi;
}
```

Open the `YDViewController.xib` file using Interface Builder and the Assistant Editor and create a simple user interface with a `UILabel` object to display the network connection status, as shown in **Figure 2**.

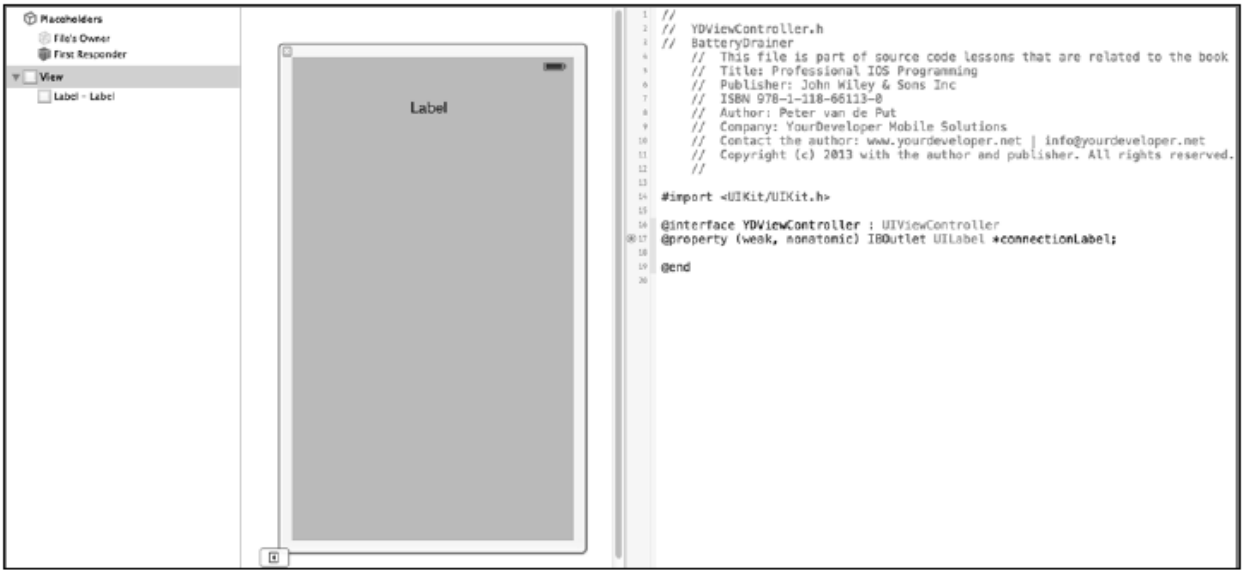


Figure 2: Create a User Interface with a `UILabel` Object

Open the `YDViewController.m` file and import the `YDAppDelegate` header file. In the `viewDidLoad` method, create an instance of the `YDAppDelegate` class named `appDelegate`. When the network status changes, it will broadcast a notification with the name `kReachabilityChangedNotification`. Create an observer for this notification targeting a method named `networkStatusChanged`.

To test the network status, you can simply call the `connectedViaWIFI` method of your `appDelegate` instance and use the return value to implement your functional logic. The complete `YDViewController.m` file is shown below:

```
#import "YDViewController.h"
#import "YDAppDelegate.h"
@interface YDViewController ()

@end

@implementation YDViewController

- (void)viewDidLoad
{
      [super viewDidLoad];
      // Do any additional setup after loading the view, typically from a
      nib.

      YDAppDelegate* appDelegate = (YDAppDelegate *)[[UIApplication
                  sharedApplication] delegate];

      [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(networkStatusChanged:)
name:kReachabilityChangedNotification
      object:appDelegate.reachability];
if ([appDelegate connectedViaWIFI])
      self.connectionLabel.text = @"Connected via WIFI";
else
      self.connectionLabel.text = @"NOT Connected via WIFI";
}
- (void)networkStatusChanged:(NSNotification*)notification {
      //you know the network status has changed so perform your action here
}
- (void)didReceiveMemoryWarning
{
      [super didReceiveMemoryWarning];
      // Dispose of any resources that can be recreated.
}

@end
```

Quick Tip

You can download the Reachability application codes directly from *https://developer.apple.com/library/ios/samplecode/Reachability/Introduction/Intro.html*.

<H2> [Battery Drainage]

Source: [Professional iOS Programming] [Chapter 15] [Page 442]

As an iOS programmer, you have to make sure your application does not drain the battery. One of the main causes affecting the battery lifetime is calling the `startUpdatingLocation` method of the `CLLocationManager` class and never calling the `stopUpdatingLocation` method.

When you download an application from the App Store, in many cases it will prompt you to authorize access to your location. This request comes from the `CLLocationManager`, which is very often used for purposes it is not intended for. Many developers create and initialize a `CLLocationManager` instance in their `AppDelegate` even if the application itself does not require it, so why do they do it?

As you will learn in the next section of this chapter, you might be interested in the location of a user to determine in which countries your application is used. With this information, you can localize your application in a certain language. When you need this insight, simply create and initialize an instance of the `CLLocationManager` class, set the delegate, and call the `startUpdateLocation` method in the `application:didFinish WithLaunchingOptions:` method.

Implement the `locationManager:didUpdateToLocations:` delegate method and store the `[locations lastObject]` value in a `CLLocation` property, which you can use for your analytical purposes. Open the `BatteryDrainer` project and add the `CoreLocation` framework to the project. Open the `YDAppDelegate.h` file and change it as shown below:

```
#import <UIKit/UIKit.h>

#import "Reachability.h"

#import <CoreLocation/CoreLocation.h>
@class YDViewController;

@interface YDAppDelegate : UIResponder
   <UIApplicationDelegate,CLLocationManagerDelegate>

@property (strong, nonatomic) UIWindow *window;
```

```
@property (strong, nonatomic) YDViewController *viewController;

@property(nonatomic,strong) CLLocationManager* locmanager;
@property(nonatomic,strong) CLLocation *userlocation;

@property (strong, nonatomic) Reachability* reachability;

-(BOOL)connectedViaWIFI;
@end
```

Now open the `YDAppDelegate.m` file and implement the `CLLocationManager` logic as shown below:

```
#import "YDAppDelegate.h"

#import "YDViewController.h"

@implementation YDAppDelegate

      - (BOOL)application:(UIApplication *)application
      didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
      self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
         bounds]];
      self.locmanager = [[CLLocationManager alloc] init];
      self.locmanager.delegate=self;
      [self.locmanager startUpdatingLocation];


      self.reachability                  =                  [Reachability
         reachabilityForInternetConnection];
      [self.reachability startNotifier];
      [self.reachability currentReachabilityStatus];


      // Override point for customization after application launch.
      self.viewController = [[YDViewController alloc]
            initWithNibName:@"YDViewController" bundle:nil];
      self.window.rootViewController = self.viewController;
      [self.window makeKeyAndVisible];
      return YES;
```

```
}

-(BOOL)connectedViaWIFI
{
  return [self.reachability currentReachabilityStatus]== ReachableViaWiFi;
}

#pragma mark CoreLocation
- (void)locationManager:(CLLocationManager *)manager
      didUpdateLocations:(NSArray *)locations
{
      self.userlocation=[locations lastObject];
      [self.locmanager stopUpdatingLocation];
}
```

<H2> [Unimpressive User Interface]

Source: [Professional iOS Programming] [Chapter 15] [Page 444]

Most iOS users expect an application to be intuitive and expressly designed for the device they are using, which might be an iPhone, iPod Touch, or an iPad. For this reason you should design your application for the device screen of the user.

Users also expect an application to respond to gestures that are familiar from other applications, such as swiping a table row to reveal a Delete button. Apple has provided a document named the **iOS Human Interface Guidelines**, which contains guidelines and principles that help you design a professional user interface.

Finally, it is imperative that you use **high-quality artwork** to enrich your application. Most programmers are not graphic designers; so to make your application successful, consider working with a graphic designer to get an astonishing user interface.

Quick Tip

You can find the iOS Human Interface Guidelines at *https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/*. These guidelines will help you design professional user interface.

Real Life Connect

Source: [iOS App Development] [Chapter 12] [Page 342]

Cartoon graphics and music help create a strong look and feel for applications, such as the mobile game Angry Birds. Although the design features are not difficult to code, a game with the same basic code features but cruder graphics would be less appealing.

Additional Knowhow

OpenGL Driver Monitor is an analysis tool that is used for gathering graphics processing unit (GPU)-related information, which includes performance data, video bus traffic, and hardware stalls. This information can tell us the reason for the slowdown, as well as to help improve the application performance.

Technical Stuff

An application should retrieve data from servers differently when the phone is on Wi-Fi and when on data (2G/3G/4G). This is because the data network consumes more battery than on Wi-Fi. It may also incur extra charges on the user. Therefore, it is wise to make the application download only the critical data from the servers when on mobile network. This can be done using the **reachability** file from the official Apple developer website.

Lab Connect

During the lab hour of this session, you will be able to perform technical analysis of an application module.

<H2> [Performing a Commercial Analysis]

Source: [Professional iOS Programming] [Chapter 15] [Page 444]

Once your application is available in the App Store and you want to monetize it, commercial analysis is an important methodology to increase your revenues. When you log in to the **Apple Member Center** via *https://developer.apple.com*, you have access to iTunes Connect, the area where you manage your applications.

When you enter this part of the Member Center, you can click the **Sales and Trends** link to view and download your sales statistics. However, the information provided is too limited to allow you to do a serious commercial analysis because it basically only displays the number of downloads per market for each of your applications.

You can also view the **In-App Purchases** per market, but the information is available only for the past 13 weeks and is discarded after that. You can download the data and process it in Excel, but you are still missing valuable information to perform a commercial analysis.

For this reason, several analytical platforms are available that enable you to perform some serious commercial analysis. It is up to you to select your analytics provider based on your specific requirements. One of the largest players in the field of gathering analytics data is the company Flurry (*www.flurry.com*), and it is free to use. It is estimated that about 50 percent of the iOS applications available in the App Store are using Flurry Analytics.

<H2> [Introducing Flurry Analytics]

Source: [Professional iOS Programming] [Chapter 15] [Page 445]

To sign up for a **Flurry account**, download and install the Flurry Analytics software development kit (SDK), and how to implement it.

When you go to the Flurry website on *www.flurry.com,* you can register yourself to create an account. Once you have verified your account, you can log in to your account and click **Add New Application** from the menu to create your application in the Flurry system. Follow the instructions on the screen to create and configure your application. Once your application is stored, you will see a screen telling you your unique application key and a link to download the latest SDK.

<H2> [Implementation of the SDK]

Source: [Professional iOS Programming] [Chapter 15] [Page 445]

Once you have downloaded the latest version of the **Flurry SDK**, navigate to the downloaded archive and within the archive navigate to the **Flurry** folder, which contains two files, the `Flurry.h` header file containing the public interface to a static library, and the library file named `libFlurry.a`.

Start Xcode and open the project in which you want to implement the Flurry Analytics SDK.

Now drag and drop the `Flurry.h` and `libFlurry.a` files into your project.

Open your `YDAppDelegate.m` file and import the Flurry header file. In the `application:didFinish LaunchingWithOptions:` method, simply implement the following line:

`[Flurry startSession:@"YOUR_API_KEY"];` replacing `YOUR_API_KEY` with your unique application key.

That's it. Each time your application is launched, statistics information is sent to Flurry, which you can then analyze on the website.

Technical Stuff

Source: [iOS App Development] [Chapter 12] [Page 345]

Xcode includes further options for more advanced developers. Unit Tests compare the value produced by the code with expected values. Scripting and Automation can feed events into an app to simulate user interaction. These powerful features come into their own in more complex projects.

Quick Tip

Source: [iOS App Development] [Chapter 12] [Page 345]

If you want to perform serious commercial analysis on your applications and you want to use Flurry, you should study the information available on its website *www.flurry.com*.

Big Picture

Source: [iOS App Development] [Chapter 12] [Page 345]

On the App Store, the bar for app reliability is surprisingly low. Many commercial apps crash regularly. Apple tests apps for basic functionality, and if an app crashes immediately, it is rejected. But some apps crash only after they have been running for some time or in very specific and limited circumstances. Apple's cursory testing does not usually find these problems, but this does not mean you can ignore these issues. Users will give your app poor feedback if it crashes too often, and it is better to make a good initial impression.

Additional Knowhow

Source: [iOS App Development] [Chapter 12] [Page 347]

Breakpoints have many powerful options. You can set them to play a sound, run an external script, or generate a log message, among other options. Most of these options are specialized and you do not need them for basic debugging, but it is useful to know that they are available.

Cheat Sheet

- Technical analysis of an application is important to ensure that users are happy and continue to use your application.
- The key technical reasons for applications to perform poorly or behave in a user-unfriendly manner are as follows:
    - Application crashes
    - Blocking the main thread
    - Memory leaks
    - Using synchronized HTTP requests
    - Extensive bandwidth usage
    - Battery drainage
    - Bad user interface
- The commercial analysis of an application will help you find the information you need to improve your application, remove functions that are never used, or localize the application in multiple languages in order to monetize it.
- Several analytical platforms are available that enable you to perform some serious commercial analysis. One of the largest players in the field of gathering analytics data is the company Flurry (*www.flurry.com*).