Chapter 5

Implementation

Introduction:

Implementation includes all activities those take place to convert from old system to the new one. The new system may be totally new, replacing an existing manual or automated system, or it may be major modification to an existing system. In either case proper implementation is essential to provide a reliable system to meet organization requirements successful implementation but may not guarantee improvement in the organization using new system.

5.1 Implementation Environment

For Implementation we have used:

- XCode 5
- iPhone Simulator 6.0 or higher
- iOS SDK 7.0
- Smart Git

GUI based Vs. Non GUI

In GUI based application it is very easy for user to understand the overall content and the previous and next step of the action. Also GUI based application does not make as issues like navigation, ambiguities in data selection and entry.

GUIs have become the established alternative to traditional forms-based user interfaces. GUIs are the assumed user interface for virtually all the systems development using modern technologies. There are several reasons why GUIs have become so popular.

- GUIs provide the standard look and feel of a client operating system.
- GUIs are so flexible that they can be used in most application areas.
- The GUI provides seamless integration of custom and package applications.
- The user has a more natural interface to applications: user understanding is improved.

Table 5.1: Comparison between GUI and NON-GUI

GUI	***NON GUI***
In GUI based there is no hierarchical ordering of forms.	In Non GUI based applications, the forms are arranged in a hierarchical order.
The most obvious characteristic of GUI application it allows multiple windows to be displayed at the same time.	With Non GUI based application we can interact with one form at a time.
There are no constraints on the order in which user may enter data on screen.	In Non GUIs, the fields on the form have a predefined and unchangeable, the user may only access the fields in a certain order, regardless of whether any data has been entered into the form fields.

GUI introduces additional objects such as radio buttons, scrolling lists, check boxes and other graphics that may be displayed or directly manipulated.	Non GUI based do not have such objects.

Table 5.2 Comparison between Single user and Multi User

Single User	***Multi User***
The major disadvantage is that when one user is accessing a system other user cannot use it.	Multiple users can access the system paralleled.

Our system is mobile-app GUI and single-user system so no parallel access of the system will be there. In our system all connections with database is not done by the process of designing. There is proper and efficient coding for connection with database.

5.2 Technology and Literature Review

5.2.1 XCode 5

Xcode is an Integrated Development Environment (IDE) containing a suite of software development tools developed by Apple for developing software for OS X and iOS. First released in 2003, the latest stable release is version 5 and is available via the Mac App Store free of charge for Mac OS X Lion and OS X Mountain Lion users. Registered developers can download preview releases and previous versions of the suite through the Apple Developer website.

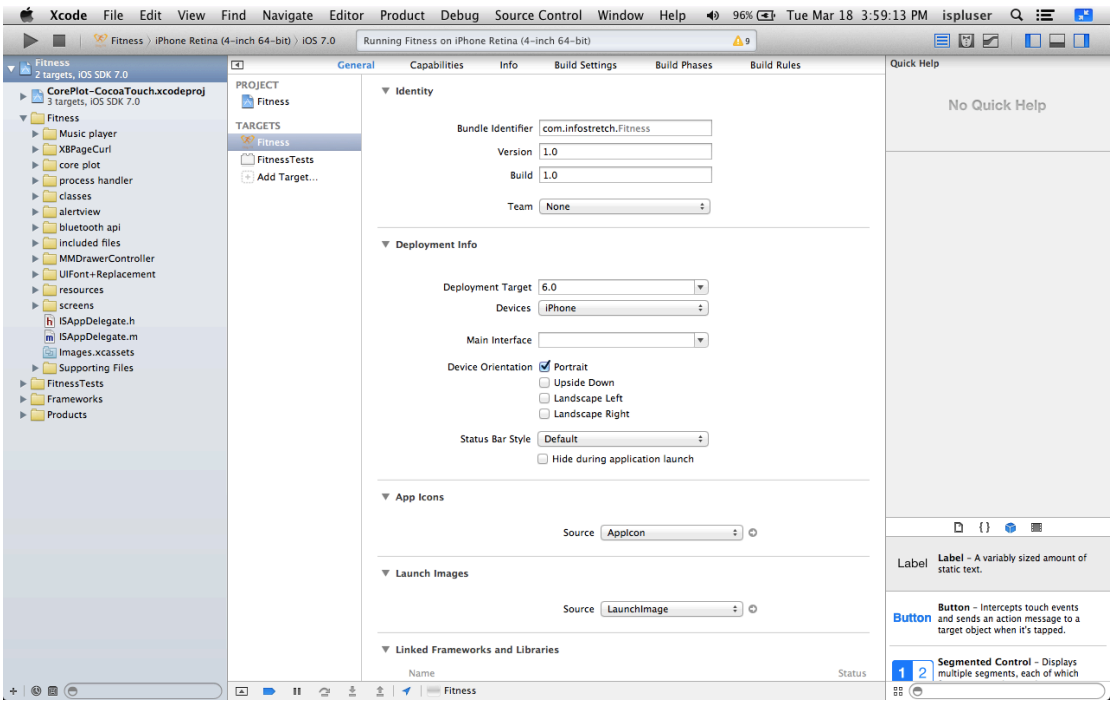

Figure 5.1: Xcode 5 Screenshot

The main application of the suite is the integrated development environment (IDE), also named Xcode. The Xcode suite also includes most of Apple's developer documentation, and built-in Interface Builder, an application used to construct graphical user interfaces.

5.2.2 iOS

iOS (previously **iPhone OS**) is a mobile operating system developed and distributed by Apple Inc. Originally released in 2007 for the iPhone and iPod Touch, it has been extended to support other Apple devices such as the iPad and Apple TV. Unlike Microsoft's Windows Phone (C++) and Google's Android, Apple does not license iOS for installation on non-Apple hardware. As of September 12, 2012, Apple's App Store contained more than 700,000 iOS applications, which have collectively been downloaded more than 30 billion times. In June 2012, it accounted for 65% of mobile web data consumption (including use on both the iPod Touch and the iPad). At the half of 2012, there were 410 million devices activated.

The user interface of iOS is based on the concept of direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. Interaction with the OS includes gestures such as swipe, tap, pinch, and reverse pinch, all of which have specific definitions within the context of the iOS operating system and its multi-touch interface. Internal accelerometers are used by some applications to respond to shaking the device (one common result is the undo command) or rotating it in three dimensions (one common result is switching from portrait to landscape mode).

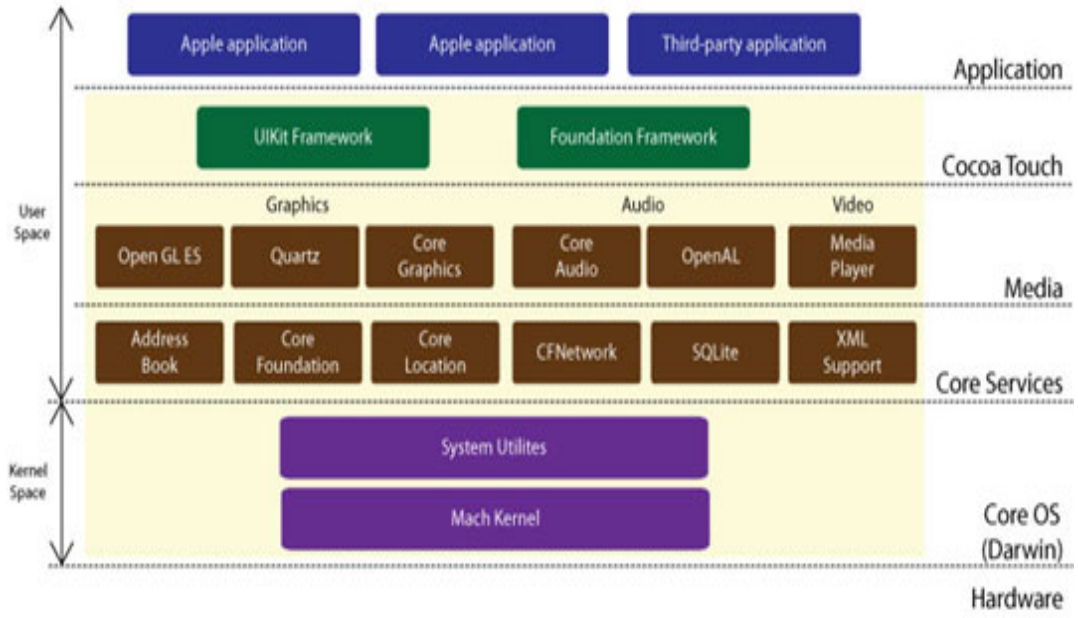

Figure 5.2: iOS structure

iOS is derived from OS X, with which it shares the Darwin foundation. iOS is Apple's mobile version of the OS X operating system used on Apple computers.

In iOS, there are four abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer. The current version of the operating system (iOS 6.1.2) dedicates 1-1.5 GB of the device's flash memory for the system partition, using roughly 800 MB of that partition (varying by model) for iOS itself. iOS currently runs on iPhone, iPad, iPod touch and Apple TV.

5.3 Modules Specification

The 'Stay Fit' Mobile Application uses objective c classes along with xib files for GUI design. It also uses xml and plist files for implementation. It is stand-alone application and is not dependent on any web service.

Module performs following tasks :

- Set Workout Goals
- Set Workout Reminder
- HR Monitor Connection
- Continuous Hear Rate Monitoring
- Undergo Workout
- Workout Report
- Music Player
- Voice Assistance

5.3.1 User Profile

This module deals with saving the user details into the application. It includes following details: Name, DOB, Gender, Weight and Height. The information will be used to calculate the calories burned during the particular workout session. This module also takes care of editing the user information and provides the other modules the details of user. It also resets and deletes all the data on user request.

5.3.2 Workout

This module incorporate the handling code for a workout and requires to use services of all other modules. It communicates with other modules time to time and collects the information about user activity. It includes following task:

- Set workout goals.
- Start new work out session either predefined or custom.
- Generate a report on the end of the workout session and also get the history of workout session.

5.3.3 Reminder

This module interacts with the native reminder application to manage custom reminder that would be directly accessible from the application. This module provides the following functionality.

- Add a reminder for workout.
- Delete an existing reminder.
- Edit an existing reminder.
- Remove an alarm from reminder.

5.3.4 BLE Connection

This module uses the core Bluetooth API and provides the support to connect and maintain the data state while data exchange. It provides the following functionality.

- Connect BLE peripheral defined according to HR profile of Bluetooth 4.0.
- Discover peripheral devices which provide the HR monitoring services
- Disconnect from currently connected device.

5.3.5 HR Distributor

This module records the HR data and provides it to other modules as needed. This module is responsible for saving the heart rate data returned from BLE module. It also plots the graph as per the requirement of the user. It briefly includes following functionality:

- Continuous Heart Rate monitoring
- Plot HR graph

5.3.6 Voice Assistance

On starting a new workout, application will start collecting workout data like, Heart Rate, Speed (Min, Max & Avg.), Energy Expended (Calories burned), Duration, Distance, Step Count, Path on Map and time to time the application will give feedback to user in form of audio. This module works in conjunction with the workout module. The workout module periodically passes the text to be spoken to this module and this converts the text data to speech.

5.3.7 Music Player

On starting a new workout, application will start Playing music which will be directly played from default music application of the device. This module communicates with the native music application of the iPhone and allows user to add the songs, which will be played during workout.

5.3.8 Social Network Sharing

Users will be able share the details of his/her workout to facebook and twitter account. The attributes of the workout will be shared on the preconfigured account of the user. This module also works in conjunction with workout module. The workout module provides the details of the workout and this module shares those on social network.

5.4 Coding Standards

Programming style is a set of rules or guidelines used when writing the source code for a computer program. It is often claimed that following a particular programming style will help programmers to read and understand source code conforming to the style, and help to avoid introducing errors.

5.4.1 General Principles

- **Clarity**

It is good to be both clear and brief as possible, but clarity shouldn't suffer because of brevity:

Example**:**

insertObject:atIndex:	Good.
insert:at:	Not clear; what is being inserted? what does "at" signify?

- **Consistency**

Try to use names consistently throughout the Cocoa programmatic interfaces. If you are unsure, browse the current header files or reference documentation for precedents.

Consistency is especially important when you have a class whose methods should take advantage of polymorphism. Methods that do the same thing in different classes should have the same name.

Example **:**

(NSInteger)tag	Defined in NSView, NSCell, NSControl.
- (void)setStringValue:(NSString *)	Defined in a number of Cocoa classes.

- **Class and protocol names**

The name of a class should contain a noun that clearly indicates what the class (or objects of the class) represent or do. The name should have an appropriate prefix (see "Prefixes"). The Foundation and application frameworks are full of examples; a

few are NSString, NSDate, NSScanner, NSApplication, UIApplication, NSButton, and UIButton.

Most *protocols* group related methods that aren't associated with any class in particular. This type of protocol should be named so that the protocol won't be confused with a class. A common convention is to use a gerund ("...ing") form:

NSLocking	Good.
NSLock	Poor (seems like a name for a class).

Some protocols group a number of unrelated methods (rather than create several separate small protocols). These protocols tend to be associated with a class that is the principal expression of the protocol. In these cases, the convention is to give the protocol the same name as the class.

An example of this sort of protocol is the NSObject protocol. This protocol groups methods that you can use to query any object about its position in the class hierarchy, to make it invoke specific methods, and to increment or decrement its reference count. Because the NSObject class provides the primary expression of these methods, the protocol is named after the class.

5.4.2 Methods

- **Accessor Methods**

Accessor methods are those methods that set and return the value of a property of an object. They have certain recommended forms, depending on how the property is expressed:
If the property is expressed as a noun, the format is:
- (*type*)*noun*;
- (void)set*Noun*:(*type*)*aNoun*;

Example:

- (NSString *)title;
- (void)setTitle:(NSString *)aTitle;

If the property is expressed as an adjective, the format is:

- (BOOL)is*Adjective*;
- (void)set*Adjective*:(BOOL)flag;

Example:

```
- (BOOL)tableView:(NSTableView *)tableView shouldSelectRow:(int)row;
- (BOOL)application:(NSApplication *)sender openFile:(NSString *)filename;
```

```
- (BOOL)isEditable;
- (void)setEditable:(BOOL)flag;
```

- **Delegate Methods**

Delegate methods (or delegation methods) are those that an object invokes in its delegate (if the delegate implements them) when certain events occur. They have a distinctive form, which apply equally to methods invoked in an object's data source. Start the name by identifying the class of the object that's sending the message.

The class name omits the prefix and the first letter is in lowercase.A colon is affixed to the class name (the argument is a reference to the delegating object) unless the method has only one argument, the sender.

```
- (BOOL)applicationOpenUntitledFile:(NSApplication *)sender;
```

- **Method arguments**

There are a few general rules concerning the names of method arguments:

- As with methods, arguments start with a lowercase letter and the first letter of successive words are capitalized (forexample, removeObject:(id)anObject).
- Don't use "pointer" or "ptr" in the name. Let the argument's type rather than its name declare whether it's a pointer.
- Avoid one- and two-letter names for arguments.
- Avoid abbreviations that save only a few letters.

5.4.3 Properties

A declared property effectively declares accessor methods for a property, and so conventions for naming a declared property are broadly the same as those for naming accessor methods. If the property is expressed as a noun or a verb, the format is:

@property (…) *type nounOrVerb*;

Example :

```
@property (strong) NSString *title;
```

```
@property (assign) BOOL showsAlpha;
```

If the name of a declared property is expressed as an adjective, however, the property name omits the "is" prefix but specifies the conventional name for the get accessor, for example:

```
@property (assign, getter=isEditable) BOOL editable;
```

In many cases, when you use a declared property you also synthesize a corresponding instance variable.

Make sure the name of the instance variable concisely describes the attribute stored. Usually, you should not access instance variables directly; instead you should use accessor methods (you do access instance variables directly in init and dealloc methods). To help to signal this, prefix instance variable names with an underscore (_), for example:

```
@implementation MyClass {

   BOOL _showsTitle;

}
```

If you synthesize the instance variable using a declared property, specify the name of the instance variable in the @synthesize statement.