Developing an iPhone application with focus on the user interface

Narayanan Kannan
Abstract

Developing an iPhone application with focus on the user interface

Narayanan Kannan

Smartphones are becoming increasingly popular day by day. The reason why they are called ‘Smart phones’ is that they can perform advanced computing and have a better connectivity to the Internet. Smartphones usually run a complete operating system. For example, iOS runs on Apple Inc.’s mobile devices like iPhone, iPod Touch and iPad [1]. A Smartphone allows the user to run applications. These applications are aimed at making life simpler for the users.

This Master Thesis aims at developing one such application called ‘Help4Dinner’ with focus on user-interface and usability. The application can make the life easier for people who want to manage the groceries they have at home and their shopping list. The interface has been carefully designed so that users can get used to the application very soon and also feel comfortable with it.

Throughout the process of design and development an iterative method of development was used. With continuous interaction with my supervisor and member of No1 Advisor, the design changes were discussed after every sprint. Testing was done at regular intervals both in device and simulator and finally the application was submitted to Apple’s AppStore on June 17th 2011. After inspection by Apple the application was published on June 25th 2011 and was available for download.
1. INTRODUCTION.................................................................................................................. 2
  1.1 Background .................................................................................................................... 2
  1.2 Goal .................................................................................................................................. 2
  1.3 Delimitations .................................................................................................................. 3
  1.4 Scope .................................................................................................................................. 3
  1.5 Overview .......................................................................................................................... 3

2. Methodology ...................................................................................................................... 4
  2.1 Knowledge Study ............................................................................................................ 4
      2.1.1 Integrated Development Environment (IDE) .......................................................... 4
      2.1.2 COCOA .................................................................................................................... 5
      2.1.3 Programming Language ......................................................................................... 5
  2.2 Software development methodology .............................................................................. 6
  2.3 USE CASE ....................................................................................................................... 7
  2.4 Competitor Analysis ....................................................................................................... 8

3 DESIGN .................................................................................................................................. 10
  3.1 MVC Design Pattern ...................................................................................................... 10
  3.2 User Interface design ..................................................................................................... 11

4 Implementation .................................................................................................................. 16
  4.1 Memory Management .................................................................................................... 16
      4.1.2 Static Analyzer in XCode ...................................................................................... 17
  4.2 Database Implementation .............................................................................................. 17
  4.3 The Application .............................................................................................................. 19
      4.3.1 The Three Tabs ....................................................................................................... 20
      4.3.2 ‘At Home’ view: ..................................................................................................... 20
      4.3.3 Add Item View ....................................................................................................... 27
      4.3.4 Shopping list view ................................................................................................. 29

5 Testing .................................................................................................................................. 32
  5.1 Testing application on device ....................................................................................... 32
  5.2 Testing for memory leaks .............................................................................................. 33

6. CONCLUSION AND FUTURE WORK .............................................................................. 34

Appendices .............................................................................................................................. 37
Appendix A ................................................................................................................................ 37
1. INTRODUCTION

This is a Master's thesis at Department of Human Computer Interaction, Uppsala University done by Narayanan Kannan from January 2011 to June 2011. The thesis will be in collaboration with the software company 'No1 Advisor', Stockholm.

The company 'No1 Advisor' recognized that many struggle with the problem of what to cook for dinner and that standard recipe iPhone apps or webpages do not help enough. The problem is that one seldom has knowledge of what groceries one has at home so that there's almost always some ingredient missing in the recipes, or you may buy extra of something that already is there. The solution is to create an iPhone app in which the users can easily manage what groceries they have at home and then search for recipes that can be cooked from what already exist at home.

The outcome of this Master’s Thesis is an iPhone application that resolves the above-mentioned issue.

1.1 Background

As described earlier, Smartphones can be considered as hand held computers that can perform advanced computing at the same time providing increased portability and space efficiency.

Apple’s iPhone is one such smartphone, which has taken the market by storm and is one of the most popular smartphones. Ever since the launch of iPhone in 2007 by Apple, they have been one of the leading manufacturers of Smartphones. With Apple providing free SDK (Software development Kit) and simulators individuals can bring their ideas to reality. As of 2011 June, the App Store had more than 425,000 applications [2].

1.2 Goal

The goal of this Master’s thesis is to develop an iPhone application according to the Human Interface Guidelines prescribed by Apple. The application should be able to run in devices with iOS 3.2 or higher.

The application would fall into the category of Household applications, since it facilitates the users to manage their groceries in their homes in an easy way. The goals are listed below.

• Design the application according to Human Interface guidelines prescribed by Apple.
• Implement the functionalities.
• Perform usability testing by handling the application to potential users.
• Submit the app into Apple’s AppStore.
1.3 Delimitations

Apple provides a document describing the human-interface guidelines that should be followed when developing an app that can be submitted to the AppStore. The document is available for download from Apple’s website [3]. Apple has stringent acceptance criteria and are renowned for rejecting applications that compete with features that iPhone provides by default. Another factor that needs to be considered when developing an app that needs to be submitted to AppStore is, Apple reject applications if there are already too many applications already in the AppStore that provide the same features.

Due to the time constraints and considering the my previous experience in iPhone development, it was decided to build an application that manage the users personal data like shopping list, ‘At Home’ list and user preferences in a settings tab and to complete the interaction with the web services to fetch the recipes in the subsequent versions.

This decision was taken after discussions with the supervisor from No1 Advisor and reviewer from Uppsala University.

1.4 Scope

The master thesis will end with a description of a working model of the application, from when it was ready for submission to the AppStore. The features in the app includes

- Ability to add/edit/delete items from ‘At Home’ list
- Ability to add/delete items to your shopping list.
- Ability to share your shopping lists with other users.
- Auto-complete entries for easy addition of items.
- Settings page where users can reset the application and store some user preferences like disabling popups.

As an Iterative method of development was followed, the design and feature lists kept changing regularly and new features were added which is described in detail in my further sections.

1.5 Overview

In chapter 2, introduces you to the environment used for development, the methodology followed for this Master thesis. In the environment used for development I give an insight about the programming language that’s been used for development. I also explain the design constraints, followed by identifying the prominent use cases. Before moving to design I also give an overview of the other applications that I analyzed in the competitor analysis.

In chapter 3, I begin by discussing the Model View Controller (MVC) design pattern used in COCOA. Then I describe how we designed the initial views of the
application, what were the design decisions that were considered, how the design changed during the subsequent iterations.

With all initial design drafts done, in chapter 4 I move on to the implementation phase. I explain how important is memory management when programming for iOS, how to properly allocate and de-allocate memory. I also explain which database is used for this application and the reasons behind choosing that. Then I move on to explain how this application was implemented in the end. I do not explain how the application was iterated in every sprint, but rather explain how the application stands as of now.

In chapter 5, I explain how we tested the application using both Static analyzer and Instruments application that comes along with XCode, while preparing for submitting our application to AppStore.

At last in chapter 6, I conclude with giving the status of the Master Thesis and the applications status in AppStore and also discuss a little about features lined up for the next version. I list the features that could be improved, added and based on user feedback.

2. Methodology

2.1 Knowledge Study

I give a brief introduction about the tools/environment that were used for the development and the programming language. Understanding these two were really important, which helped me in the later stages of development.

2.1.1 Integrated Development Environment (IDE)

Xcode is a set of tools provided by Apple for developing applications to iOS and for Mac OS X.

Xcode consists of

- **Xcode IDE**: It consists of a GCC Compiler, a powerful text editor, a debugger and an iOS simulator. Xcode is available for download for free, after registering in Apple’s developer account. For my thesis, I used Xcode 4.0 with Mac OS 10.6 [4]

- **Interface Builder (IB)**: It is a part of Xcode suite and is very useful for creating graphical user interfaces. Interface Builder provides objects like tables, buttons that the COCOA developers can easily drag and drop to create the user interfaces [5].

- **Developer Documentation**: It consists of Apple’s Human Interface guidelines, sample codes, how to’s, tutorials etc. The main idea behind
these documents is to ensure users have a consistent user experience and behavioral experience across applications.

2.1.2 COCOA

Cocoa is the application-programming interface (API) used for building software programs on iPhone, iPod Touch, iPad and Mac OS X. Cocoa Touch, an extension of Cocoa exclusively for the above mentioned devices other than Mac OS X provides support for touch screen gesture recognitions and animations.

The main features of Cocoa touch are

1. Animation
2. Multitasking
3. Touch screen gesture recognizers [6].

2.1.3 Programming Language

Apple uses a specific programming language for developing applications for Mac OS X and iOS called Objective-C. It is a superset of C, which extends the object-oriented capabilities. All of the syntax for non-object-oriented operations (including primitive variables, preprocessing, expressions, function declarations, and function calls) is identical to that of C, while the syntax for object-oriented features is an implementation of Smalltalk-style messaging [7].

The syntax is quite different from other object-oriented programming languages. Hence to understand the code snippets that I will be showing, I will give a brief introduction to the language.

Data types:
All the basic data types in C language are supported by Objective-C. In addition to it there is a distinct data type called id, for any kind of object regardless of the class.

$id$ data;

This means that we can store object of any class in $data$.

Method Declarations:
There are 2 types of methods in Objective-C. Class methods and Instance methods. Class methods can be called without any instance of the class but instance methods, as the name says it needs a particular instant of the class to call the method.

General syntax of method declaration is

$\text{scope}($returnType$) \text{methodName:}($parameterType$)\text{ParameterName}$;

Example:

$-(\text{void})\text{searchTable:}($NSString$ *)\text{searchText}$;
Here the “-” indicates that is an instance method. “searchTable” is the method name and “searchText” is the argument of type NSString.

-(void)setName:(NSString *)name withID:(NSInteger)rollNo;

This is the example of a method with more than one argument.

**Method Passing:**
In Objective-C a method is called by sending a message. For example,

```c
[student setName:foo withId:2];
```

We call the method “setName” and pass two arguments “foo” and “2” to some object “student”

**Class Instantiation:**

A class can be instantiated by allocating memory and initializing it. For example,

```c
Student *student=[[Student alloc]init];
```

The `alloc` message ensures the object is allocated enough memory to hold all the instance variables of that object.

**2.2 Software development methodology**

**AGILE METHODOLOGY:**

Agile methodologies are typically used when the requirements of the product is difficult to freeze and document initially. Hence it allows the developers to shift their primary focus on product itself, rather than designing and documenting at every stage. They breakdown the tasks in small increments and does not focus on long term planning. Each iteration ends with the complete cycle of planning, designing, coding and testing. Iteration may not add enough functionality to warrant a market release, but the goal is to have an available release (with minimal bugs) at the end of iteration [8].

The biggest advantage of agile methodologies is they cut down development costs and focus on getting the product to the market soon. Since the work is done in iterations its easier to prioritize the requirements for the subsequent iterations. Agile methods are considered to be the opposite of sequential methodologies like Waterfall Model.
Figure 1. In waterfall model the stages of analysis, design, coding and testing are separate and documents are written and frozen after each stage. However in Agile model, each increment has all these stages. [9]

For this thesis, I decided to use Agile Methodology [9] because it was very difficult to freeze the applications requirements in the beginning. Also the design of the individual screens had to be iterated based on feedback. The requirements were mainly gathered from analyzing what a user needs and also from a competitor analysis by checking out other applications that belonged to similar genre. Since new applications or new versions were added to AppStore very frequently, we decided to go with a minimal set of requirements in the beginning and add requirements in subsequent iterations.

For example, a feature called "Quick Add" items to your list, which I will explain in detail in the later stages, was added towards the end, as we felt it would make the first time users of the application to get easily going.

2.3 USE CASE

A use case is a series of steps that can happen between the user and the software product [10]. They help in determining the requirements and also prioritize them. It is extremely useful in projects where the basic requirements should be listed in the beginning but new requirements will be added as the project grows.

For my application the use cases are scenarios between the users of the application ‘help4Dinner’ and the application itself. The following use case diagram lists all the primary Use cases that were decided before the detailed design of the application.
Figure 2. Use case that I formulated for Help4Dinner application

- **Manage ‘At Home’ and Shopping list**: Basically this is the core idea of the application. A convenient interface with which they can manage both the lists.
- **Add/Edit/Delete Items from ‘At Home’ list**: Populate grocery items as when users buy. Edit items incase they want to increase/decrease the quantity. Delete item when they empty the item.
- **Search using Search bar**: When users populate with too many items, they can search for items using a search bar.
- **View Items based on categories**: Since a grocery item can be stored in pantry/fridge/freeze. Users can collectively see all the items they have stored in one particular storage place.
- **Add/Delete from Shopping list**: Users can create a shopping list before they go shopping.
- **Move items to ‘At Home’ list after buying**: This again is one of the core features surrounding the main idea, where users can easily move all the items to their ‘At Home’ list after they are done shopping.

2.4 Competitor Analysis

Competitor analysis is the process of analyzing the strengths and weaknesses of similar products in the market [11]. It helps a great deal in understanding the product, how competitors have designed it and also read comments about their product. It also gives an immediate warning if we have ignored some key features in the design or if a feature is not that popular in the market.
Since we realized that there were lot of other applications that could manage shopping lists, and also since we wanted an application that is not so different from other applications, we decided to study all our competitor applications. The applications were carefully chosen and a long list of all applications was noted down. The list of all applications analyzed is listed in Appendix A.

The analysis for each tested app started with downloading it from the AppStore and installing on my device. All the user comments that the application has received in the AppStore were also noted. Once the applications were installed I used the application as a customer after reading the description. As and when I used the application the most prominent features, the strengths and weaknesses of the application were written down.

Since all the applications were downloaded from AppStore, they all satisfy the human interface guidelines provided by Apple [3]. This helped me a lot in designing the application in detail. Some features were very common in all competitors’ applications and hence were added to the list of requirements. For example most of the applications have tried to cut down the time user spends in adding a new item to their list. To facilitate this they have ‘autocomplete’ feature which matches the characters that user enters to the inbuilt database of items and displays a list of the items based on that. In that way users do not have to key in the entire item name. The competitor analysis helped me identify all these small things that can make the application more usable.

One another feature that I had missed in my requirement analysis phase was bar code scanning to enter items into the list. Apparently we believed developing this might take longer and decided to skip this feature until the next versions. In general most of the applications had a sectioned viewing of the items, sharing your shopping list, which I had already included in my set of requirements.

We also had to choose between certain features, as we felt the feature would not help the overall idea of getting the recipes based on what we have at our ‘At Home’ list. Features like tracking the expiry dates, indications if the items were running low in your pantry were not considered for the design. Both these features will require the users to be continually feeding the information or else the application is going to give them wrong information. After a small survey with my colleagues we decided not to have those features, as not many preferred it.

Another important thing I learnt from the competitor analysis phase was to keep the design as intuitive as possible. For example I noted that applications, which requested the user to do a ‘shaking gesture’, were criticized in the user comments. Also I realized that the importance of making the user to input as minimal information as possible. Since I repeatedly entered my ‘At Home’ list in various applications, I felt it would be really nice if all units, quantity and storage fields were prefilled. This was one of the key ideas that I gained from using the application over and over. I explain how this modified my design in the design phase.
3 DESIGN

COCOA [6] uses an MVC design pattern.

3.1 MVC Design Pattern

The Model-View-Controller design pattern splits the application into three independent sets of interacting classes. The reasoning behind MVC design pattern is to break the complex application into smaller parts to simplify the process.

Before I go on to explain the advantages and why COCOA uses MVC design [20], I explain what these three different objects can do.

Model object:
The model object usually consists the data encapsulating the instance variables needed for the object. It is highly reusable and it contains an initialization method to return instances to various other classes. In short a model object contains the applications data and logic. In my Help4Dinner application an example of a model object would be the Item class.

```swift
@interface Item : NSObject {
    NSInteger iId;
    NSString *iName;
    NSInteger iQty;
    NSString *iUnit;
    NSInteger iStorage;
    NSInteger iNeed;
}
```

The Item class inherits from superclass NSObject. It also consists of a set of instance variables to store the data. The class also contains a set of accessor methods for these instance variables (which I have not shown in the code snippet). The class also contains custom methods to modify the data, but this code snippet is just to explain the concept of a model object.

View object:
View objects are used to display the data of the model object. Everything that a user sees on the iPhone screen is a view object. The fundamental role of the view objects is to display information on screen and respond to user interactions such as touch and input events.

As explained earlier XCode provides a tool called Interface Builder, which can used to design user interfaces and test the feel of how it looks on the iPhone screen. Interface builder consists of a library of COCOA touch objects that can simply be dragged and dropped on to the screen to design the User interface. In my application I have a number of views to display various information on the screen.
**Controller object:**
Controller object acts as the intermediary between the model objects and the view objects. They are responsible for calling the appropriate methods in the model for updating and are also responsible for providing the view with appropriate data to display.

The following diagram would help in understand the concept of Model-View-Controller better.

![MVC design pattern](image)

MVC pattern makes the objects extremely reusable since they are divided and are independent of each other. They also help a great deal in extending the program. Since the views and the models are decoupled now, it helps in easier understanding of the concepts and development becomes easy.

### 3.2 User Interface design

From the knowledge I gained from using similar apps in competitor analysis stage, and from keeping in mind the use cases and requirements for 'Help4Dinner' application I made the first draft of the user interface design.

The user interface is the view in the MVC model explained earlier [chapter 3.1]. The user interface was designed after removing the unnecessary features that requires attention and also with a focus of keeping it simple and elegant. Since the primary motive of this Master Thesis is to submit the app to AppStore Apple’s Human Interface guidelines were also considered.

I initially designed 4 views for my application.
1. 'At Home' view
2. Edit/Add Item view
3. Shopping list view
4. Settings view

I decided to use the same view for Edit Item and Add Item view, as they would have the same fields (item name, item quantity, item unit, item storage place.)
‘At Home’ view:

I made a rough sketch of the ‘At Home’ view in PowerPoint. This view will be the default screen that will be displayed to the users when the application is always launched. The user can toggle to Shopping list and Settings view directly by using the tab bar at the bottom. This tab bar will always appear as long as the application is active, to enable easy transition to other views.

This view displays all the items a user has already added to the list as shown in Figure 3a and 3b. The unit and quantity field are shown in braces for each item. There is also a disclosure button (>) at the end of each item, to notify there is another view, which will show up on choosing the button. From this view users could easily navigate to the ‘Add Item’ view by choosing the add button to the top right corner in the Navigation bar. A search bar was placed just below the navigation bar. Below the list of all the items, a tab for categorical viewing was added. The tab bar had 4 buttons, 3 for categorical viewing (pantry, fridge and freezer) that is if a user chooses one of the categories say fridge, the list will display only those items that are stored in fridge alone. Using the sort button, users can choose between how they want the list to appear from the following two options

- Sort by Alphabetical Order (default)
- Sort by Input Date

The users could also delete an item directly from this view, using the left swipe gesture that is common in iPhones. This is shown in Figure 3b. Using the left swipe gesture brings up ‘Delete’ button for that particular item and clicking on that deletes the item from the list. And also the item is automatically added to your Shopping List. Hence your Shopping List automatically grows when you start using the application.
As I mentioned earlier, since we used an agile methodology, design suggestions were taken in after each iteration review and the corresponding changes were carried out in the subsequent iterations. Hence the final view will be a little different from what we see here with a few more features to enhance usability.

‘Add Item’ view:

Add Item view will be used a new Item to the ‘At Home’ list and also to the shopping list. I decided to use the same view for both the list, so that the users will not have too many different views to confuse them.

This view was considered to be the most important, as the real user input is going to be in this view. The user might not like it if he was made to touch the screen too many times. There are text fields for name, quantity, units and buttons below them, for the users to select where they will be stored. This button input is used for categorical viewing in the ‘At Home’ list. To make the input process faster we decided to add a list that contains items that matches the current characters that the user has entered. This idea is similar to the search list that appears when you start typing in an URL in your browser. Clicking on the list automatically fills in the corresponding item name in the ‘Name’ text field. This idea was later extended to have a prefilled list of all the items fetched from a database and have all the fields such as units, quantity, and storage place to be automatically filled when the user clicks on an item. I explain this feature in detail with an example as I come to the implementation stage. The point to be noted here is that we decided to have that feature, so that users can enter items even faster than manually entering each field. This is one feature that makes our application different from others in the market right now.
Figure 4 shows the initial design of the ‘Add Item’ view.

![Figure 4a. Add Item View](image1.png) ![Figure 4b. Edit Item View](image2.png)

The navigation bar for this screen had a ‘Save’ button on the right and a ‘Cancel’ button on the left. Clicking on the ‘Save’ button saves the item into the list (‘At Home’ list or Shopping list depending on from where the Add Button was chosen). Clicking on ‘Cancel’ button aborts the adding process. Both the screen returns the control to their parent view.

Figure 4b shows the Edit item view. The edit item view is shown when the user clicks on an Items name or the disclosure button (>) from the ‘At Home’ list. When doing so, the edit item comes up with corresponding item name, unit and quantity prefilled which the user can edit and then save. Figure 4b shows the edit view, when the user has clicked on Milk from the ‘At Home’ list.

**Shopping list view:**

The Shopping list view is shown when the user chooses the second tab on the tab bar of the application. This view shows all the items that the user has listed in the Shopping List already.

This view is similar to the ‘At Home’ list in a lot of ways. For example, the navigation bar has an Add button which when clicked shows up the ‘Add Item’ view. Similarly users can use the left swipe to delete the items from the shopping list as well. Perhaps the difference between the Shopping List and the ‘At Home’ list would be the different buttons in the bottom toolbar and the difference in behavior when the user clicks on an item. When the user clicks on an item in the Shopping List, a tick mark (✔) appears on the right hand corner of the item
indicating the user that they have already bought the item. In addition to it was decided that the item should automatically be moved to the bottom of the list, so that all the un bought items are at the top and all the items that were bought remains at the bottom. Clicking on a bought item again brings it back to the top.

As discussed in the user requirements sharing of shopping list is very necessary. Hence we add a button ‘Mail to’ which when clicked opens the default mail application with the users shopping list already filled. All that the user needs to do is to enter the email address of the recipient to share the shopping list. The other buttons in the toolbar are for deletion and moving the items to ‘At Home’ list when they users are done shopping.

Clicking on the ‘Delete’ button provides the users with two options
- Delete all the items from the list
- Delete only the bought items.

Clicking on the ‘Move To Stock’ button again provides the user with two options
- Move all Items
- Move only bought Items

Figure 5 Shopping List

These 4 headings are self-explanatory and needs no further explanation. Figure 5 shows the initial design of the Shopping List. Not all the features that were explained here are shown in the screenshot, and cannot be shown also due to the space constraints, but its good to show the overview of the ‘Shopping List’.
**Settings view:**

The initial design of the settings tab was pretty simple. All that we here an option to set user preferences in the settings tab and some special feature like ‘Resetting’ the application. This button when clicked can reset both users’ shopping list and ‘At Home’ list. We believed this feature was important, so that users can empty everything before they go for a vacation.

I am not attaching any design image of the settings view, as I had not finalized it during the design phase. As a matter of fact, the settings page was totally different from what we had in mind. Hence I will explain about the design phase in the implementation phase.

**4 Implementation**

There were several implementation issues that occurred during the development, and I will discuss and walkthrough these issues in this chapter.

**4.1 Memory Management**

One important issue that needs to be understood and addressed before starting with iOS programming in Objective-C is an important topic called Memory Management. In section 2.1.3 I explained how to allocate memory and initialize the objects. Objective-C provides two memory management techniques.

- Reference counting
- Garbage collection

The second method, Garbage collection is a new feature that was introduced in Objective-C 2.0 where the system manages what objects are needed, what objects are currently held by other objects and frees them accordingly. This is supported only for Mac OS X development and hence I will not discuss this method.

In Reference counting the developer needs to keep of track of the objects and free memory accordingly. Every object has its own reference or retain count. When the object is allocated, the reference count of the object is set to 1. And since the retain count of the object is greater than 0, Cocoa understands that some other object needs this object and does not do anything about it. When the object is not used anymore, the developer needs to decrease the retain count and when the retain count 0, Cocoa sends a dealloc message, to free the memory. This memory can be then used for allocating other objects. To be precise somewhere in the program code, there should be a release message for every object that’s created or is retained earlier.

For example,

```objective-c
NSString *name=[[NSString alloc]initWithString:@"foo"];
//the retain count of name is 1.
[name release];
//retain count of name is 0
```
It is also important to understand that *release* does not de-allocate the object. It decrements the *retain count* only.

It is a must to have this basic understanding of memory management, before going to the implementation as memory leaks can sometimes be very difficult to track down. Since the aim of this Master thesis, was to submit an application to AppStore, one of the primary conditions to be satisfied is the absence of memory leaks in the code. Hence memory management was really considered important. There are two ways to check for memory leaks. One is the Static Analyzer that comes along with XCode (from version 3.2). Another option is to use the Instruments application. I explain in detail about Instruments application when I come talk about testing the application. We look at static analyzer now as it has the ability to identify memory leaks even before out program is run.

### 4.1.2 Static Analyzer in XCode

Unlike traditional compiler warnings, the XCode 3.2 static analyzer has a much deeper understanding of your code. By traveling down thousands of possible code paths, the analyzer identifies logical errors such as unreleased memory—well beyond simple syntax errors normally found at compile time [12].

### 4.2 Database Implementation

The internal database in iPhone is very important for this application. The application should be able to store the various grocery items the user has in their house in a table (‘At Home’ list table) and store various items that they are willing to buy in a separate table (Shopping list table). There are two different database options that can be used for iPhone application development.

- Core Data
- SQLite

Core Data is a wrapper over SQLite and has a few advantages over SQLite. But Core Data was introduced only after iOS 3.0. Hence users with iPhone OS lesser than 3.0, will not be able to use the application. Since the target users of my application can be iPod touch users with OS lesser 3.0 also, I decided to go for SQLite. There is another advantage of using SQLite, it is more portable than Core Data. For example, if at a later stage we want to develop the application in other platforms like for a PC, or for Android OS it becomes much easier when using SQLite.

I use SQLite3, a free database management system that comes along with the iPhone SDK. The SQLite3 that comes with the iPhone SDK does not have any user interface installed. Hence I used the SQLite manager Add-On for Mozilla Firefox to create my databases. Following picture shows the screenshot of the SQLite manager in Firefox.
The first step in implementing the application was designing the database. I created three different tables for my application to manage the data easily. One for the 'At Home' list and one for the Shopping List and the last one for the pre-filled items. As I mentioned earlier, since we wanted to reduce the time, that the user spends in adding new items to the list, we came up with a comprehensive list of the most common items used and entered it in the pre-filled table. The data from this table was used for implementing the Auto-Complete suggestions feature.

The Shopping table (slist) was created using the following SQLite command

```
CREATE TABLE "slist" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "name" VARCHAR,"quantity" INTEGER, "unit" VARCHAR, "store" INTEGER, "need" INTEGER DEFAULT 1)
```

Each shopping list item contains 5 attributes

**id** is the primary key for the table, which will be indexed and automatically incremented. Having a primary key makes updating and referring to easier.

**name attribute** is the name of the item (eg Milk). This name attribute appears in the shopping list. Similarly **quantity** and **unit** are associated with the shopping item.

**store attribute** of integer type is used to determine where this grocery item will be stored after being bought. In my application a storage value of 0 indicates the item will be stored in pantry, 1 in fridge, 2 in freezer.

**need attribute** is used to monitor if the item is still needed. The need value of 1 indicates the item has not been bought yet. When the item is bought, the need value is changed to 0 and updated in the database.
This database file is saved with a .sqlite file name. This was added to my XCode project using Add Existing files option under the Resources category. Next thing was to add the library to the frameworks in XCode. This was done by right clicking on frameworks in XCode and select Add/Existing frameworks and then choosing libsqlite3.dylib.

4.3 The Application

With the initial design of the application and basic understanding of the framework, Objective-C I began the process of converting the design and features into an actual application. As I mentioned earlier we used an agile method of development, hence after every sprint, iPhone users from No1Advisor tested the application. During the initial few weeks the application was tested in the simulator that comes alone with XCode, but once we registered for an Apple ID in Apple’s developer website we were able to import the device on our iPhones. I discuss how we went about testing the application in our devices in section 5.1. The periodic testing of our application in both simulator and iPhones let us identify lot of key issues that could enhance the usability of the application. Some designs were changed, functionalities were added, and some features where even removed. I am not going to explain what design changes happened, what was completed in each iteration rather I would rather explain how the applications stands as of now after all the changes.

Clicking on the application icon, on devices in which the app has been installed, can launch the application. Figure 7a shows the screenshot of the application icon and how it looks on an iOS. The application icon (figure 7b) was carefully designed according to the Custom Icon and Image Creation Guidelines section from the Human Interface Guidelines provided by Apple [13].

![Figure 7a Help4Dinner application](image)

![Figure 7b Application Icon](image)
4.3.1 The Three Tabs

As mentioned earlier, in order to provide seamless transition between the different features, a tab bar was used with three different options. The tab bar is always visible throughout the life cycle of the application. The different options in the tab bar are

At Home: This view is where the user stores all the items they own in their pantry. Users can add/edit/delete items in this list. This tab is the default selection and hence the view associated with this tab will be visible whenever the application is launched.

Shopping List: This view is where the user can write down what items they need to buy when they go shopping the next time. The design of the previous tab makes it really easy to add items in to this list. There is a red circle with a number indicating the number of items they have in their shopping list. This is called the Badge Value. This is like a notification to grab attention of users to indicate they still have items to buy, when they are in other tabs.

Settings Tab: In this tab users can set user preferences. More details on what settings can be set will be explained as I describe each and every tab in detail.

![Figure 8. The Tabs in 'Help4Dinner' application](image)

Due to the time constraints and the lack of designer in our team, the images you see for the individual tabs were purchased from an external source. They provided support for both iPhone 3GS and the iPhone 4. One of the requirements from apple is that in order to support iPhone 4, we should include retina display images along with the resource bundle. Since the external source [http://glyphish.com/](http://glyphish.com/) provided both the formats, we decided to purchase them by paying a small fee.

4.3.2 ‘At Home’ view:

This view will be the first to be seen by the user, as it is associated with the first tab. The ‘At Home’ view is similar to the design shown in section 3.1, but the following feature was added during the life cycle of the project.

Quick Add Feature:
Consider this scenario, a user downloads the application for the first time. All that he sees is an empty list in the ‘At Home’ list. Assuming that every user will have more than 5-6 items in their house, the users has to add items one by one in the application. This might be pretty annoying and the user might not add all the items. Also consider this scenario, a user goes shopping and has bought all the items he needs. Now in order to populate his ‘At Home’ list, he needs to add
these items one-by-one again which might consume time and might be frustrating. Hence we decided to add this feature called “Quick Add feature” where users can select multiple items at one single go. The primary challenge in adding this feature was we had to decide on what were the most common items that users buy. And also other challenge was, apart from selecting the common grocery items, we should decide on the most likely unit, quantity and place of storage. For example, if a user selects Milk from the ‘Quick Add List’, it should add Item name: Milk, Quantity: 1, Units: Liter, Storage: Fridge. Creating this kind of list was difficult, and we after reading numerous articles we created the best that we could. Still we were not uncertain about what units to use for certain items. For example, Potatoes could be bought in kilograms and also as units. But we decided to address this question, after receiving feedback of version 1.0 from users. A comprehensive list of around 50 items were created and added to the database. The basic idea was to make it easier for the users to add lot of items. Figure 9 shows the screenshot of the “Quick Add feature”.

![Figure 9 Quick Add View](image)

In order to make it convenient for the user to scroll, the list was sorted based on the storage place (Pantry, Fridge, Freeze). Within these categories, grocery items were sorted in alphabetical order. As shown in the figure, as and when the user selects an item a tick mark appears towards the right of the item name, to indicate that the user has already chosen the item. In order to deselect an item, user needs to click on that item name again. When the user is done selecting the items they can select ‘save’ button on the top right hand corner of the navigation bar to add the items to the items to the ‘At Home’ list. Choosing the ‘cancel button’ empties the selection list and returns back to the first tab view, the ‘At Home’ list. At present users cannot add items to this list, but its perhaps something that we should work on in the next version. This “Quick Add feature”
was added at the fag end of the project. These use cases were found when the application as installed in 3 iPhones of people from No1 Advisor team. Hence we decided to have a pop up appearing when the application is first installed from AppStore, asking if the user wants to use the quick add feature to populate their list easily. Figure 10a shows the popup that appears only for the first time that application was launched (not every time). I check this by using the NSUserDefaults class. The class provides a programmatic interface for interacting with the defaults system. The defaults system allows an application to customize its behavior to match a user's preferences [14].

```swift
int x = [[NSUserDefaults standardUserDefaults] integerForKey:@"FIRST_TIME"]; if( x == 1 ){} else{    [[NSUserDefaults standardUserDefaults] setInteger: 1 forKey:@"FIRST_TIME"];    [[NSUserDefaults standardUserDefaults] synchronize];    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:nil message:@"Would you like to use our Quick Add Feature to populate your 'At Home' list?" delegate:self cancelButtonTitle:@"No, Add later." otherButtonTitles:@"Yes",nil];    alert.tag=1;    [alert show];    [alert release]; }
```

First I read the integer value for the key, if it is not equal to 1 then I display an alert box and also change to 1, so that the condition fails the next time. Choosing ‘Yes’ in the popup opens the Quick Add View. Choosing ‘No, Add later’ closes the popup and just shows an empty ‘At Home’ list.
However we also realized the “Quick Add feature” should be accessible even otherwise, and hence we decided to have Quick Add accessible always as the last cell in the list as shown in Figure 10b. Hence that way user can use the “Quick Add feature” to add a list of items anytime into their list.

Figure 10b also clearly shows how the ‘At Home’ view looks. This view has a navigation bar at the top with a 'Edit' button to the left and ‘+’(add) to the right. Just like the design a Searchbar was added below the navigation bar. The lists of items in the list are displayed in alphabetical order in a UITableView [15]. Each item has a disclosure button ( > ) to its right to indicate that users can edit the item by clicking on the cell. Just above the tab bar there is a tool bar, which contains buttons for both categorical viewing of the list. For example, if user chooses Fridge, the table shows only the items that are stored in the fridge. By default the application shows all the items in the ‘At Home’ list. There is ‘Organize’ button towards the right of the toolbar, which when clicked shows up a UIActionSheet as shown in the figure 11a below. The design was kept simple so that even if the user uses the application for the first time, they should be able to figure out what this view can do.
Figure 11b shows how users can search using the search bar. Note how a cancel button is added to just next to the search bar. When the user clicks on the search bar, a keypad automatically pops up, which users can use for typing characters. The application is also programmed in such a way that if the user chooses to view items stored in Fridge and then uses the search bar then it checks over the items kept in fridge alone and not the entire list.

Now, when the user clicks on an item name, it goes to the update view for the item. Update view is the view where user can change the item name, increase or decrease the quantity, change the units and storage place. This view is important since the items added using the ‘Quick Add list’ comes with the specifications that we entered. Hence users can quickly add the items to their list and then can change the specifications when thy have time at their own leisure. The ‘update item’ view comes up with the item name, quantity, unit and storage field already selected. So the users can change only that they feel that needs to be changed. Figure 12 shows the update view for ‘potato’. The update view has a lot of changes since the design made in section 3.2, but I will explain these features a little later in the ‘Add Item’ screen.
At last I will explain how to delete an item from ‘At Home’ list actually works. Deleting an item can be done using two ways.

2. Using Left Swipe.

Left swipe to delete is standard gesture in most iPhone applications. Left swiping on a UITableViewCell shows a delete button on the right corner of the cell. This is shown in figure 13b. Choosing the ‘Edit’ button on the top could also do deletion. Clicking on the ‘Edit button’ brings up a red button to the left of each shopping item with a standard iPhone animation. Clicking on this red button again brings up a delete button. Figure 13a and 13b shows the steps involved in deleting an item and how exactly it looks in an iPhone.
When does a user delete an item? When he is out of stock, or the item has expired. In either case the user might want to add the item to his shopping list in most cases. Adding the item directly to the shopping list would also annoy, incase the user never wanted to add the item to his shopping list. Hence we decided to give both the options to users once they click on delete from 13b. The options on pressing delete are shown in the picture below.
Choosing ‘Yes’ from the menu options deletes the item for ‘At Home’ list and adds the grocery item to the Shopping list. Choosing ‘No’ deletes the item and does not add to their Shopping list. Choosing ‘Cancel’ does not delete the item just closes this options menu.

4.3.3 Add Item View

The Add Item View is accessible by pressing the ‘+’ button form the Shopping List and the ‘At Home’ list. In this view users can add items, which are not there in the Quick Add list. The advantage with this view is that, users can enter their specifications to the item when they add unlike the Quick Add Feature where the items are added with the specifications that we entered in the database. This view was first designed and the same view was retained for edit item view as well. As you can see in Figure 12, the view has undergone a lot of change since the design. First the navigation bar on the top has two buttons, ‘Save’ to the right and ‘Cancel’ to the left. The ‘Save’ button appears only after the user has entered an Item name, so that even by mistake a user does not add an empty item. Clicking on ‘Save’ adds the item to the item to the list depending on from which user came from. For example, if the user presses the ‘+’ button from his Shopping list and adds an item, the item is added to the shopping list, similarly for ‘At Home’ list as well. The ‘Cancel’ button closes this view and goes back to the view where they came from.

The Add Item View has a text box for entering the item name, and quantity. Its programmed in such a way that when the user enters the name, keypad with alphabetical characters is shown and when entering quantity a number pad is shown making it convenient for the users. As mentioned earlier during the design, when I was asked to enter all the items in my fridge I had to add 5 items. For these 5 items I had to add the quantity, units and storage place as well. Hence a mandatory 4 clicks per item, plus the keypad and number pad typing. It was then when we decided to come up with this idea, which will probably make our application stand out from the rest. As and when the user starts entering the item name, we have table showing the closest match of the item name with what the user is typing.

This is shown figure 15a, when I try to add “Milk”. The suggestions start coming up as I start typing the first character and keeps changing as and when I key in more characters. The suggestion list is meanwhile scrollable, so the users can enter the first few characters, scroll and choose the item they wanted. The idea, which makes this application stand out is that, as a user selects the item name from our list of items using the suggestions list, the application not only fills the name of the item, but all the other specifications like quantity, unit and storage place automatically. So all that the user needs to do, is to change the specifications that they need to, thereby reducing the number of times the user needs to touch the screen. For this the most challenging part was to come up with a comprehensive list of items along with its details and at the same time considering the applications memory warnings.

27
For most of the items it was easier to fill in the details like Milk, which is generally bought in Liters and stored in fridge, but for other items there were more than option like Banana, which can be bought in Kilograms and also in units. We realized it was impossible to come up with a perfect list. However it is possible for the users to edit the item before saving as well. Figure 15a and 15b shows the process of adding ‘Milk’ to a shopping list.

![Image of iPhone screens showing item details and options for quantity and units]

Figure 15a & 15b. Shows the process of adding ‘Milk’ by choosing from the suggestions list. Choosing ‘Milk’ automatically fills in the other details form the database.

Now if the user wants to change specific details of the item that was added, they can do so by using the buttons on this add item view, which becomes more like editing an item now. The ‘less’ and ‘more’ buttons on the either side of quantity makes it really simple to change the quantity number. Using these additional buttons users can directly increase/decrease the quantity. Otherwise, user has to press the quantity text box, which will bring up the number pad with which they need to delete the previous number and key in the new number. This will obviously take more clicks; hence we decided to add these two buttons. Now the quantity of milk can be changed to ‘2’ with just one click.

Changing units might not be relevant in this case for milk as its mostly liters, but for other items, users can change the units by using the UIPickerView [16]. It is more like a spin wheel with a number of options. For this application we decided to have kg, grams, liter, dl, package and can as the options to choose for the units. The users can also change the storage place by just tapping on the 3 options.

Another change that was made since the design in section 3.2 is the ‘Save and Add more’ button to the bottom of the screenshot. When tested on the device, we
realized that in order to add an item the user has to press the ‘save’ button in the top right hand corner of the navigation bar. Clicking this button adds the item to the list and also closes this ‘Add Item’ view. Hence if the user wants to add another item he has to click on the ‘+’ button again to get to ‘Add Item’ view again. This might frustrate the users once they are familiar with the application and when they want to add a list of items to go press save to see the view go, and then press ‘+’ again to add one more item. Using the ‘Save and Add more’ button users can add an item to the list, without closing the view. Once pressed the item is added to the list and the screen is refreshed instantly to have the name, quantity and the units field blank to be ready to add another item. These kinds of changes were made possible only because we adopted an agile method of development, as we had usability checking of the application after every sprint.

4.3.4 Shopping list view

All the views that we have seen so far were accessible from the ‘At Home’ tab. “Shopping list” is the second tab in the tab bar. Figure 16 shows the screenshot of this view, which might look exactly like the ‘At Home’ view in the first impression. However this page was again designed carefully, keeping in mind what users generally want to do in a shopping list screen.

![Figure 16a Shopping list in 'Help4Dinner'](image1)

![Figure 16b](image2)

The navigation bar is same as the one in ‘At Home’ list with two buttons, which also does the same functionality. The shopping list is displayed in a UITableView and the last cell of UITableView is always an option to use “Quick Add feature” to a number of items at one go to shopping list.
Perhaps the main difference would be the way in which the shopping list behaves when the user selects a row. In ‘At Home’ list when users clicks on a rows, it navigates to the edit view for that item. However in shopping list, selecting a row, means the user has purchased that item. The item is grayed and is moved to the bottom; thereby always the un-purchased items stay at the top. For example if I purchase Honey and Rice, I tap those rows once and they are grayed and moved to the bottom as shown in figure 16b. Also note how the Badge value (the circle in the tab bar is changed to 1).

The shopping list has a toolbar, with a different set of buttons:

**Trash:** The title/image is self-explanatory. Clicking on trash displays an option to users asking if they want to delete all the items from shopping list or just only the ones that has been purchased.

**Email:** Clicking on email, opens the default mail application of iOS with the un-bought items

**Move to:** When users are done shopping, we realized there should be an easy way to add the items to their ‘At Home’ list and hence this button. Clicking on this button displays an option asking users if they want to move all the items or only the purchased items. Figure 17 shows how the different toolbar buttons work.

![Figure 17. Screenshots when each toolbar button mentioned above is pressed.](image)

### 4.3.5 Final Changes

With the Shopping List view and the ‘At Home’ view, the major part of the application was done. I explain the settings page in the next section, after describing what were the flaws we found with our design, when we tested the application after these two views and functionalities were done.

We realized that it was possible for the user to add the same item again and again which will make the data redundant and no checking was done to prevent this. For example if a user already has Potato in his ‘At Home’ list he can still add another item ‘Potato’ and no notification was given to users to inform that it
already exists. Whereas the correct thing would be to edit the item ‘Potato’ and increase the quantity in users ‘At Home’ list. We felt this was a usability issue if the users were not notified and added a feature that can check if the item exists in their list already. Figure 18a shows how an alert box is displayed when the user uses ‘Add Item’ view to add an item that already exists. Figure 18b depicts how users are notified when they use a ‘Quick Add feature’. In this case, if the users add 4 items out of which just one item already exists, the remaining 3 items are added to the users shopping list and the count of number of items added is shown in the notification.

![Alert boxes to notify users when they add item that already exists](image)

**Figure 18a & 18b Alert boxes to notify users when they add item that already exists**

### 4.3.6 Settings Page

As mentioned earlier Settings page can be used by the user to set user preferences. During the design phase we had not decided on what preferences a user could choose but after all the features completely, We did a test run and decided on what preferences we as a user would like to set.

In section 4.3.2, I explain how deleting an item from ‘At Home’ list works. User is shown with three different options to do with that item, delete to add it to shopping list, just delete and cancel the screen. While we did the test run, we found it was annoying to have a pop-up appear every time an item was deleted. Hence we decided to have a setting to turn it off. To be short, when the preference is set ‘ON’ user will get options of what they want to do when deleting. When it is turned ‘OFF’ the item is just deleted, no options are shown.
Similarly pop up is displayed when the user adds an item using the ‘Quick Add’ list, displaying how many items were added and note reading, existing items will not be added. This could also be frustrating after a user gets used to that system. After using it for a while, the user will get familiar with the application and it becomes needless to have a pop up. Hence another user preference setting was added. Figure 19 shows how the settings screen appears in the final version of the application.

![Settings View](image)

Figure 19. Settings View

As you can see there are three other buttons. Reset application was an additional button we added towards the end of the project. As the title of the button suggests, using this button users can empty both the Shopping list and the ‘At Home’ list. This might be really useful when the user wants to go on a vacation or if he has not updated the application for sometime. Instead of manually emptying the list, this button could reset the application completely. The feedback button was added to take in user feedback after the application was launched. Clicking this button opens the default mail application with our email address and subject already filled. Users can just enter the body of the mail and send us some feedback. A screenshot of how this mail application looks is already shown in figure 17. The Facebook fan page button can be used to do some advertising, about what features the users can expect in the next version of the application and create some user group and maybe a poll before we decide on what to be done.

5 Testing

Testing here refers to testing that was done before the application was submitted to AppStore. As mentioned in the previous section in-house usability
testing was done at regular intervals after every sprint of implementation. However to submit to AppStore we had to test for memory leaks and also check if the application crashes at some point like failing to fetch from database, empty table etc.

5.1 Testing application on device

In order to check if the application crashes, the application had to be imported on to iPhone or an iPod. Testing in the simulator is the first step, but is not sufficient. For example testing with Wi-Fi and 3G can be done only with device. I am not going to explain how to import the application on the device, as there are so many resources available on the Internet. Apple’s documentation is also very good in this regard. [16] Especially, it provides detailed step-by-step instruction about how to deploy the app in phone. While running on the device, the application did not crash and hence this condition was satisfied and good for submission from AppStore.

5.2 Testing for memory leaks

Lots of importance was given to testing for memory leaks and testing was done using Static Analyzer (Section 4.1.2) and another comprehensive tool called Instruments. Instruments shows a time line displaying any event occurring in the application, such as CPU activity variation, memory allocation, and network and file activity, together with graphs and statistics [17].

Figure 20 shows a screenshot of the Instruments application checking for memory leaks when the application was run. Though no leaks are shown in the screenshot, initially I did have some leaks but with the help of stack trace in instruments I was easily able to identify the leak and fix it.
Figure 20. No memory leaks were detected using Instruments application

When this was done, we decided the app was ready for submission. Submitting to AppStore was a tedious and a complex process. However with the help of Apple’s documentation I managed to submit it [19](However you need to have a developer apple account to view these resources).

6. CONCLUSION AND FUTURE WORK

After 7 long days, Apple reviewed the application and approved it. The application was available for download all over the world from June 25th 2011. As of now when I write this report the application has more than 400 downloads a majority of which are from United States and Sweden. The process of creating such a comprehensive without any background in iPhone development was pretty tough but with the numerous resources available on the web, I was able to implement it successfully. The fact that the application was accepted by Apple the first time itself, tells that we had followed the Human Interface Guidelines. During this journey I realized the importance of design and usability. I became familiar with XCode, Objective-C and iOS development. I might not have been able to explain all the features in the application; hence if you own an iPhone I would recommend you to download the application from the link below.  
http://itunes.apple.com/tw/app/id444935034?mt=8

As mentioned earlier, the application was divided to accommodate with the time span allocated for a Master thesis. The primary task from now would be integrating the application for the overall idea of fetching the recipes. The user can just create one shopping list in a device now; perhaps in the next version
users should be able to view a common shopping list with their wife or their friends.
REFERENCES:

Appendices

Appendix A

Applications that let’s users manage both ‘At Home’ list and Shopping List:

MyFridge
Itemshelf
Pantry
In My Pantry
GrocerEaze
StillTasty
List to Pantry
Fill the Pantry
Best before
iFroze
MiPantry
Home Food Storage

Applications that let’s users manage only Shopping List:

ShopperLite
ShopListFree
GroceryZen
Shopshop
Shoppinglist
ShopSavvy
ShoppinglistLite
BaM Classic
Shopper

In Swedish:

Liztor
Inköplista
Handla
MarknadLista

Other Swedish related apps:

ICA Maxi
MatHem
Recept (Tasteline)
Livsmedel
Arla köket
15-min recept
Coop recept
Recept lite
Barilla kokbok
Kokaihop.se
Mat & dryck (Meny.se)