A study of the iOS

An exploratory article on how large of a role the iOS has played in the success of the iPhone

Kristian Sandström
Abstract

The iPhone has become a top selling smartphone since its launch in 2007 and has along with its iOS (Apple’s mobile operating system) overtaken many popular and established mobile phone brands in the ratings. Some competitors have not been able to provide a competing smartphone while others have grouped together to withstand the Apple onslaught.

There are probably quite a few reasons why the iPhone has become such a success, and one could likely write a report solely about those reasons. I will focus on one of them in this study, maybe one of the most important reasons: the iPhone came with a revolutionizing user interface. I have studied iOS applications and their user interface components along with related research in order to try and find out what exactly makes it so attractive and also to uncover any flaws I might stumble across along the way. In order to answer this, and to gain a better insight into the iOS, I have developed an iPhone application from scratch covering all of the basic functionality usually found in any other iPhone application. The results will show that most of it seems to relate with Apple enforcing very strict rules and guidelines, along with limitations placed on the developer and the process as a whole. This strict process ends with Apple evaluators performing a review of the finished product, using their guidelines as heuristics. These guidelines will be proven to have scientific credibility, and the controlled development process will be a key to defining the iOS success.

1. Introduction

1.1. Background

According to market share reports from the first quarter of 2011 (Flores, 2011), Apple has surpassed Nokia and taken the number one spot on the European charts with just over 20 percent. This is counting smartphones only, since Apple only sells the iPhone. The chart supplying these numbers compares smartphone developers. Another popular comparison seen in a lot of charts is categorizing and comparing the number of smartphones sold by their software platforms. On this list, Android OS has the largest market share worldwide followed by Symbian, an OS maintained mainly by Nokia (Savov, 2011). Apple iOS places third on this list. What is very important to remember is that iOS is unique to Apple devices - the iPod touch, iPhone and iPad - while Android and Symbian can be found on a multitude of different smartphones. This means that while iOS is not the leading mobile operating system, the iPhone still outsells each individual smartphone equipped with Android or Symbian (Dalrymple, 2011).

Since its release in 2007, the iPhone has obviously become very popular. That may be thanks to a large Apple fan base, clever marketing, the revolutionary touch display or maybe a mix of them all. All of these reasons and probably many more all add to the fact that Apple has become a prominent telephone developer with only one telephone, and a quite high-end one at that, on the market. Some giants in the industry like Nokia or Ericsson that have been around for over 50 years and have a large series of telephones, both smart- and feature phones\(^1\), on the market, now have to play catch-up with Apple to stay in the race for the smartphone market.

But that is not to say Apple hasn’t had any obstacles in their way. One topic of discussion has been the fact that the iPhone lacks a physical keyboard or numpad. One article (Senia, 2007) describes a disappointed audience which noted that writing an SMS took twice as long on the iPhone compared to their normal regular input telephone. Another topic of discussion was the fact that the iPhone 1 lacked 3G support. Apple defended themselves with the fact that the 3G technology was still underdeveloped and caused a serious drain on the battery. 3G was implemented a year later with the iPhone 3G. There were also speculations

---

\(^1\)Term used to describe telephones with less computing ability than smartphones.
about the fact that while iPhone was successful in the US, sales might look very different in Asia and Europe, where touch screen devices and smartphones were old news.

What might have been the biggest deterrent for many potential customers was the price. There were two versions of the first iPhone; you were given the option between either 8GB or 4GB storage. They were priced at $599 and $499 respectively, which meant that they were quite a bit above the $100 to $300 price range of most other popular mobile phones of the time (Aquino, 2007, and Wikipedia A). Like that wasn’t enough, customers needed to add the cost of a carrier contract as well as the monthly fee. This figure could equal another couple of hundred dollars in the US. Apple would address this problem in September 2007, when they cut the price of the iPhone 8GB by a third and discounted the 4GB version.

Closely tied to the pricing issue was the fact that the iPhone was bound to a single operator. This caused quite some problems when the iPhone was launched outside of the US, where for example Australia, Germany and France took the issue to court, claiming such a lock was against their countries laws. A Slovenian operator started selling "unlocked" iPhones without any official contract with Apple, causing quite some confusion.

Except for the lack of a physical keyboard or numpad, none of the above complaints had much to do with the graphical interface of the iPhone. This lack of complaints about the interface suggests that it may be a large reason for why the iPhone became so popular in such a short period of time, and provides the base of my research question.

1.2. Research question & aim

The primary research question of this article is: What makes the iOS such a success seen purely from a software point of view? Is it a revolution in interface design or are there flaws to be discovered under the surface? I will discuss this and try to back my findings with theoretical support.

I hope to gain a clear insight into this one reason behind the iPhone rise to power, and clarify whether it’s all a master plan from the Apple design department or if there is a certain amount of luck, or calculated risk, involved as well.

My study will focus on the iOS and iPhone. I won’t discuss iPod or iPad, since the only noticeable differences are screen size and the fact that you can’t make a phone call from those devices. I will try to avoid going into details about other software platforms, and although that might yield some interesting results, I’m prevented by lack of access to such smartphones. I believe that I will still produce a sufficient answer to my question through my methods of choice.

2. Methods

I chose a qualitative approach to my research. A qualitative approach suits my purpose well, since the traditional way of conducting a qualitative study is comprised of three steps known as Analytic induction (Hartman, 2004). Step one is planning your study, step two is gathering information and the final step is analyzing it. This means that a theory isn’t formed until you have gathered and analyzed your information. A quantitative method on the other hand is usually performed the other way around, where you formulate a theory first and conduct the information gathering and analysis afterward.

My planning phase resulted in a form of to-do list, where I first needed to explore logistical solutions, learn app development, make the app, gather literature and look at current research, analyze it along with my app and write the report. My step two included gathering information while making the app, reading literature and studying articles. My final step would be writing this report and ultimately answering my research question with a theory resulting from my analysis.
2.1. iPhone Application
I accepted a practical assignment from The Chimney Pot – a post-production agency in Stockholm. Their CEO, Henric Larsson, wanted me to make them an iPhone application intended for themselves, other professionals in the industry, and their customers. The app (short for application from here on) was to include some various features which I realized would cover most of the standard functionality seen in other apps. Since the app would include several functions and pages, I knew I would run into a lot of practical problems along the way. I would note these issues and give myself a crash course in iOS app development along with smartphone interface design in general. When I had finished, I would use all the experience and information gained from the process along with a heuristic evaluation of the product using Apple's own design principles.

2.2. Heuristic Evaluation
By applying Apple's own design principles to my project, I hoped to tie my little project to the big picture, which in this case would be the iOS in general. Since there's been more than 10 billion downloads from the app store (Apple B) and experts predict 100 million units sold by the end of 2011 (Paul, 2010), we get a rough average of 10 app downloads per device. This means that iPhone users do a lot more with their smartphone than making phone calls or surfing the web. If all these apps are made following the same design principles that were applied to the iOS itself, and users spend most of their time in some sort of app, I would like to propose that the apps are blended with the iOS, and one might not necessarily evaluate them differently. This means that evaluating my own app with the Apple design principles will equal into evaluating the iOS platform, hence giving me the ability to answer my research question.

2.3. Literature
A large source of information for my study comes from websites. There are two main issues with this (Falgas et. al., 2008), first; finding the actual references for the facts on websites can be difficult. Sometimes, people just state what they know, without any reference as to where they learned it from. And sadly, people are often mistaken, or even make things up. Popular informational websites such as Wikipedia\(^2\) are built by visitors, but Wikipedia has managed to maintain credibility thanks to an enforcement of references to whatever facts are stated. Unsupported facts are usually presented with a warning depicting them as such. The second issue is the "Page not found" problem. Websites go offline, are remade and are updated. It is therefore important to choose sources carefully, and if possible always try to find the reference that the source used. I have done my best to always confirm facts by looking up multiple trustworthy sources and referencing the appropriate one.

I will try to balance this with research literature and articles, although sometimes, literature searches don't return anything usable, forcing me to look at websites. But to the degree that it is possible, especially when concerning research related facts, research references will also be provided.

3. Related Research
Since my qualitative methods won't yield a theory before the end of this report, this section contains information on general theory and terms concerning my study; such as usability.

\(^2\)Http://www.wikipedia.com – An open, non-profit, community driven fact-sharing website.
3.1. Evaluating usability

Usability falls under the HCI (Human-Computer Interaction) field of research and is largely an on-going discussion, especially as new devices with new interfaces are continuously developed (Blanchard, 1998). Usability doesn't have one clear definition, as Donyaee and his colleagues (Donyaee et. al., 2006) point out, since it is applied to such a broad range of products and uses.

Measuring usability is often done with a heuristic evaluation. A heuristic evaluation is performed by one or more evaluators along with a series of checklists, mental or written. Options to a heuristic evaluation are empirical, formal or automatic evaluations (Nielsen and Molich, 1990). While the formal method is still being widely researched and not put to much actual use, and the automatic option should never be used on anything large-scale, the empirical evaluation is the only viable alternative. An empirical evaluation is performed by a test group monitored by one or more experts. As Nielsen points out through an empirical evaluation is quite costly in terms of time and expertise required, from both tester and test group. Since I happen to lack both of these, I have chosen to perform a heuristic evaluation.

One major down-side of this choice is that one person's point of view will only cover so much. Nielsen gives an approximate 20%-51% problems found when a heuristic evaluation is performed by only one expert. Still, problems can be large and minor. For my study, I am looking for any large and obvious flaws with the iOS interface. I am not trying to identify every minor issue that my finished app might have, such as a button being slightly too big or small. Minor problems like those are irrelevant to my discussion since they are customizable and fixable. What I’m looking for are those things that aren’t as easily fixed, problems that are a thorn in the side of the developer but mainly the end-user. I believe these problems are also those that are quite obvious, especially given the time that I in the end will have spent with both developing and evaluating the app.

3.2. Heuristics

Usability criteria can range up to the thousands. It is however highly impractical and time consuming to examine something at so many points. Usability criteria can also, like the definition of usability, vary. Nielsen proposes a compressed list of 9 heuristics, which are displayed along with Apple’s human interface design principles (Apple A) in table 1 below. As mentioned earlier, I will apply the latter to my finished app in the evaluation process. It is therefore important that I verify the integrity and discuss each Apple criteria.

<table>
<thead>
<tr>
<th>Nielsen</th>
<th>Apple</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Simple and natural dialogue</td>
<td>Metaphors</td>
</tr>
<tr>
<td>2 Speak the users language</td>
<td>Reflect the users mental model</td>
</tr>
<tr>
<td>3 Minimize user memory load</td>
<td>Managing complexity in your software</td>
</tr>
<tr>
<td>4 Be consistent</td>
<td>Consistency</td>
</tr>
<tr>
<td>5 Provide feedback</td>
<td>Feedback and communication</td>
</tr>
<tr>
<td>6 Provide clearly marked exits</td>
<td>Modelessness</td>
</tr>
<tr>
<td>7 Provide shortcuts</td>
<td>Direct manipulation</td>
</tr>
<tr>
<td>8 Good error messages</td>
<td>Explicit and implied actions</td>
</tr>
<tr>
<td>9 Prevent errors</td>
<td>Forgiveness</td>
</tr>
<tr>
<td>10</td>
<td>Perceived stability</td>
</tr>
<tr>
<td>11</td>
<td>WYSIWYG</td>
</tr>
<tr>
<td>12</td>
<td>Aesthetic integrity</td>
</tr>
<tr>
<td>13</td>
<td>User Control</td>
</tr>
</tbody>
</table>

Table 1: Nielsen vs. Apple heuristics
As we can see, Apple has 13 criteria in their list. Here is a summary on how Apple defines their criteria, along with a comparison at each criteria and how they relate to Nielsen’s heuristics:

- **Metaphors:** Use people’s knowledge of the world and concepts. This usually translates into the use of icons for linking and association. This is closely related to Nielsen 1 & 2 (see table 1). Speaking the users language and using a simple and natural dialogue is a summary of a large amount of minor criteria, and can be applied to many of the following Apple heuristics.

- **Reflect the user’s mental model:** People expect things to work in a certain way. When typing an e-mail, you write your message, choose a recipient, and send. This is again related to Nielsen 2. Speaking the users language and designing for what the user expects, are the same.

- **Managing complexity in your software:** All about reducing clutter and making your interface more streamlined and easy to use. As the software becomes more complex, so does the interface have to become easier to understand. This can be coupled with Nielsen 9. What Apple wants you to do boils down to preventing errors by making the interface understandable and easy to use.

- **Consistency:** Throughout the individual program, but also across programs. Meaning Apple wants your app to look and function like other apps, as to avoid confusion and make the app feel trustworthy. Nielsen 4, being consistent.

- **Feedback and communication:** Not only implies providing error messages but also to continuously provide information about what is going on at every step that the user might take. Related to Nielsen 3 and 5 through 8. Nielsen 3 suggests minimizing user memory load, this can be achieved by providing good feedback in your app and reminding what the user of choices made and any information input. Also includes providing steps back in a process, through back buttons, undo actions or time lines.

- **Modelessness:** Modelessness, the lack of a model, implies making the interface and navigation as free as possible. The user should never be locked in a process or trapped on a certain page, but should always have the alternative to leave and go somewhere else as she wishes. Nielsen 6 & 7; providing shortcuts and clearly marked exits.

- **Explicit and implied actions:** Instead of remembering a series of commands, actions should preferably be listed in menus or similar. An explicit action is shown in such a menu, depicting an action that can be taken on a particular object at a particular time. For example, bringing up the options menu on an item, and clicking the "Delete" option. Implied actions on the other hand are triggered by visual cues and tied to the user’s mental model. Dragging something to the garbage can is an implied action, where it is assumed that the dragged object will be deleted. This practice can be tied to Nielsen 1-3. Speaking the users language in the form of actions doing what the user expects it to. It is also a big part of minimizing memory load of the user, where he doesn’t need to remember how to do each action manually when there is a simple menu of actions to select from.

- **Direct manipulation:** Describes the degree of control that the user has over objects on screen. The most common example of direct manipulation is the drag-and-drop operation. If a text is small, one should expect some sort of easy manipulation to zoom in. Direct manipulation is one form of implied action, therefore connecting to Nielsen 1-3.

- **Forgiveness:** Similar to both modelessness and feedback, forgiveness suggests making the app free from terminal errors. A user should never be afraid of clicking a button or following a link. Again, providing ways to step back in the process and ways out of it. Nielsen 5-9 all touch upon this.

- **Perceived stability:** Usage of common interface objects and actions. The app should appear to be in line with other apps in reference to this. The user should feel a familiarity in the interface, and have control over things she is used to controlling. Closely tied to Nielsen 4; being consistent.

- **Aesthetic integrity:** Basically the measure of good visual design. Apple suggests keeping clutter to a minimum and going for the clean design concept. It also includes matching graphics with their meaning, such as check boxes presenting options while radio buttons force an exclusive choice. Nielsen doesn’t
present a criteria clearly evaluating the aesthetic design. As it has always been, aesthetics are somewhat subjective, and beauty is usually in the eyes of the beholder. Still, Apple only suggest an overall aim and layout, they aren’t trying to tell you how to bend curves or gradient your background. Heuristics are first and foremost meant to evaluate the design of an interface. The aesthetic values are slightly different and must be evaluated as such (Kumar and Garg, 2010).

- **User control:** Don’t make software where the computer takes control from the user. The designer should assume that the user knows what she is doing and keep "Are you sure" messages to a minimum. If it seems needed that the software starts performing actions that the user could be interested in doing, the interface probably needs a redesign. This is related to Nielsen 2,5 & 8, speaking the users language in the form of making a interface that works with the user. Providing feedback and good error messages at moderate and appropriate times.

- **WYSIWYG:** "What You See Is What You Get". Where a user is permitted to lay something out or design something themselves, the output should be no different from the input. The interface shouldn't hide features from the user, but present what it has to offer in a smart way. This is Nielsen 3 and a somewhat inverted number 6. The user shouldn’t have to remember what he wrote in a text box, as it should presented unchanged on the next step in the process. With inverted Nielsen 6, I mean that WYSIWYG suggests providing clearly marked entry points as well as exits.

Summarizing we see that the Apple guidelines are closely related to the Nielsen heuristics – giving them a scientific integrity and enabling use of them in an evaluation. The Apple guidelines are somewhat targeted at Mac OS X (iMac operating system) and iOS software, since that is what they develop. This doesn’t affect my evaluation since iPhone apps fall under the iOS category.

### 4. Results

#### 4.1. Defining a "good" user interface

Something that requires some attention is how we define a good UI (short for user interface from here on). iPhone being a revolutionary, unique, innovative device is all good and dandy, but what does it mean? Hearing all these fancy words automatically make us assume that the UI is good and perhaps better than its predecessors. We usually forget that these descriptive words may as well describe something in a negative way. But I want to stop and look at the cause for all these claims, discussing them with the previous chapter about heuristics in mind. Apple did indeed revolutionize the smartphone when they removed all physical input except for one single button. This multifunctional button acts mainly as a back button, always returning you to the device start menu or desktop. It also wakes the device from sleep mode as well as functioning as control for switching off, switching between, moving or uninstalling apps. Instead of some sort of tilt or flip function revealing a hidden keyboard, the iPhone provided an almost all-covering screen. This was indeed new and unique, therefore the descriptive terms may apply. This does, on the other hand, not mean that the UI is good, since it only concerns the hardware of the device. This brings us to the next topic: Apple introducing the first multi-touch display.

A multi-touch display is a touch surface that can recognize and follow more than one point of contact. Multi-touch technology research began in 1982, and a company called Fingerworks developed multi-touch devices from 2001-2005 before being acquired by Apple (Wikipedia B). Apple went on to say they invented and patented the multi-touch as part of their iPhone marketing campaign (Carlson, 2010, Apple D), which is clearly false. The fact that they were first to introduce it on a smartphone is not. Again, this mostly concerns hardware, what the actual software does with the opportunities enabled is another story. As we know, iOS put this hardware to good use, making the pinch-gesture zoom a standard. The iOS gains a point for being first with this feature, adding another layer of interaction to their device.
One final hardware feature, that again was not invented by Apple, or first on a mobile phone for that matter (All About Symbian, 2006), is the accelerometer. An accelerometer is a sensor that can detect movement in a certain direction. The iPhone was the first smartphone to integrate the accelerometers in a more practical and interface related way. Prior mobile phones that included accelerometers used it mainly as a gimmick, providing a game or two showcasing the functionality but not really putting it to greater use. The iPhone UI on the other hand would come to set another smartphone standard; displaying content on the screen, mainly images and video, appropriately when the screen is tilted. Application developers are encouraged to use all of the features provided, leading to a widespread development around the accelerometer functionality. Apart from games there are many useful applications like the variations of a crash detection utility, which detects dramatic g-force input and automatically dials emergency services. Another point for the iOS for seeing the potential of a hardware feature and naturally integrating it into the software.

So it seems that the descriptive words about the device aren’t just made up for marketing purposes. But what about the design of the actual UI? We can conclude that the UI at the time of release presented several features which enhanced its value, but a good UI should look nice and actually provide the required functionality as well, right? Design research suggests that images are superior to text when it comes to visual search and item finding (Kurki et. al., 2002). The iPhone UI is mostly made up of images representing actions, navigation and other links. Other strong interface criteria include easy manipulation and direct feedback. With all the functionality packed into the iPhone it took a leap ahead in terms manipulation and direct feedback. Combining the large, touchable interface with sensors such as accelerometers made the interface feel organic, it seemed that you could manipulate the UI in any way you wanted and it would respond as expected. An organic interface is what other research suggests as a reason for the iPhone success (Schwesig, 2008).

One final part of the iPhone that made the UI stand out, was ironically enough it’s lack of features (Dredge, 2008, Gentner and Nielsen, 1996). The popular trend of mobile phone development before 2007, was going bigger and bigger, including more and more functionality and gizmos. Apple went the opposite direction. The iPhone was stripped of a 10 megapixel camera, GPS, 3G, carrier freedom of choice, innovations in music playing or voice recognition, even a physical keyboard was removed. After launch, the internet forums and question boards were filled with topics regarding problems with making actual phone calls with the iPhone. Making a Google search today for any combination of "iphone phone call problem" will return approximately 9,100,000 results (yes – 9 million). Some experts even went as far as to dub it "The worst telephone in the world" (Graham, 2009). As astounding as all this sounds, it didn't seem to matter. Reading some of the threads will show a faith in Apple, in most answers someone would encourage the complainer to be patient and wait for a firmware update. Surely the problems would be solved, and anyway, what did it matter when you had such a nice gadget to play around with? The world approved of the UI so much that it actually didn’t matter that the iPhone was a bad telephone, when it had so much else to offer.

With all these facts, and without even having to go especially deep into any design heuristics, I think it can be concluded that the iOS provided a good UI. And as an end note; it shouldn’t be forgotten that the iPhone has gone through 4 years of development since it’s release, with customers awaiting the iPhone 5 sometime in September 2011 according to www.iphone5release.org.

4.2. The Chimney App

4.2.1. Developing the app
Before I had any idea how iPhone apps are actually developed, what coding language is used etcetera, I thought my background in website development would help me. I was pretty wrong. Apps are first of all software, not web applications, meaning it's developed in an entirely different coding language and practice from what I'm used to. There are things such as memory handling and more complex variable defining
compared to web languages. Although knowing any code language will of course lessen the challenge of
learning another, I had to face quite a steep learning curve the first couple of weeks. For reference, iOS
development is done in Apple’s developer tool Xcode. The coding language is called Cocoa, which is an
extension of Objective-C, which in turn is an extension of C. As the days passed and things started making
sense, I started learning quite a lot about how apps are made, and why a lot of them seem to look the same.

I shouldn’t forget to mention that The Chimney Pot first had to buy a developers license for $99 before
any development could begin. You need to buy this license even if you are going to make and distribute free
to download apps. There’s a more expensive enterprise edition of the license, costing $299 per year, which
allows you to distribute your apps in-house with ease. The only free solution is to apply for a university
developers program, where a qualified, degree granting, higher education institution may receive a
customized license (Apple E).

After receiving this license, you download Xcode. Xcode is basically a code editor and interface builder in
one. In the interface builder, you are provided with a visual representation of the iPhone screen, on which
you drag and drop interface components such as text labels, buttons and images. In the code editor, you then
connect these visual elements to code functions. There is also the option to build the entire app purely
through code, although the beginner may find it way easier, and faster, to go with the other approach.

Dragging and dropping may sound somewhat amateurish, but actually works very well. Although what
surprised me was that Apple provided all the components. Along with these components you are provided
with suggestions, and limitations, on how to use them. While working with my app, I ran in to a lot of
problems, that in the end turned out weren’t actually problems but restrictions forced by Apple. One such
problem is for example width and height restrictions. Most elements have a minimum or max limit on both,
making customization just as limited. Another is the basic functionality built in to certain elements. For
example, a text box allows the user to click said text box and the virtual keyboard will pop up. What bugged
me for some time was the fact that the "enter" button didn’t work. Although I was done typing, the keyboard
didn’t go away when I asked it to. As it turns out, there is no done button built into the keyboard, the
developer has to make his own done button and add functionality to present it with the keyboard, and then
connect functionality to the button that forces the keyboard to close when clicked. You can customize the
keyboard, but when I looked for ways to do this I quickly decided it was way to complicated. That is why in
most apps you see, there are always "Done" buttons on the navigation bar in the top right, or elsewhere on
the interface.

Many more issues like these exist, and the reason for most of them can usually be found in the Apple
developer documentation (Apple F). Here, Apple describes each component along with it’s intended
purpose. And indeed each component has a very specific purpose. All these issues that I encountered weren’t
just random bugs in Xcode but very thought out specifics set in place by Apple. The minimum and maximum
height and width specifications are there to prevent a button being too small for a finger to easily click, or a
picker (the slot-machine resembling component) from taking more space than there is on screen. Apple also
points out that the developer is responsible for deciding when and how the keyboard is dismissed, although
no reasoning behind this is provided. As a developer I don’t have a problem with this, it’s easy enough to do
via code, but the point is that Apple says and developer does. There could as well have been an option to add
a done button via the interface builder, but there’s not and that’s just the way it is. The whole program is
structured in this way, with many components having an predefined Apple purpose, and using the
component for another purpose will prove hard and sometimes impossible. The fact is, Apple doesn’t really
want you to make any new custom components unless it is absolutely necessary Many of the Apple design
principles (see 3.2) recommends using the presented components and visual elements. The following quote
can be found at the bottom of the Apple design principles (Apple A):

“Be very cautious about creating new interface elements because you may introduce unnecessary
complexity. You will have to work extremely hard to make sure that any newly introduced elements fit
in with those provided by Cocoa and Carbon. Additionally, as the Aqua user interface continues to evolve, your custom elements will require updating to adapt to changes in Aqua.”

So as it is, Apple prefers us to use their framework as much as possible. According to their guidelines, this is to keep the overall feel of stability and familiarity across the platform. My earlier statement that the high use of apps somewhat blend them with the iOS is given some credibility here. This quote is also found among on the Apple design principles page:

“People rely on the standard Mac OS X user interface for a consistent, predictable user experience.”

Reading the whole page will give you an indication that Apple really wants you to design apps that will be associated with the general app and iOS look. I’m not saying this is necessarily bad, but designers may be somewhat limited in their creativity, especially when we move on to the next section of this report, discussing the Apple submission process.

As my work continued, I didn’t note any more major disturbances or oddities. Most of my time was spent working out solutions for various coding issues. After about a month, the app was more or less finished. It now included:

✔ A start screen displaying the company mobile website. The initial goal here was to make a custom RSS reader, but the challenge of making one combined with the fact that it wouldn’t look nearly as good as the mobile website led me to discard that idea. You’re not allowed to use the Apple RSS reader in your app either, or to configure it in any way. The end result requires an internet connection and uses a technique for displaying websites in your app.

✔ An informational section, containing pre- and post-production related facts on green screening, motion tracking and a glossary. This part of the app uses tables and its accompanied navigational system. This is one of the common techniques often found in iOS apps, as shown in image 1.

✔ The perhaps most interesting, at least most challenging part to develop, was the data rate calculator. The user specifies video frame size, frames per seconds, frame bit depth and may add audio or a compression. Based on this data, the user may enter a certain number of frames, film duration or disk space to get the respective values. Example: User chooses 1280 x 720 pixels, 25 frames per second, 16 bits per pixel. This equals approximately 138 megabytes per second. If the user specifies a duration of 10 seconds, the calculator will return a total frame number of 250, a total disk space requirement of 1,38 gigabytes, as well as the length, meters or feet, of physical film required to record the video, depending on film format. A detail of the calculator interface can be found in attachment 1.

✔ The last tab consists of some general information about the company and the app. Looking at image 1, it may seem like I’ve done little. This is not true, but represents the fact that the somewhat limited ability to customize the app interface will lead to it resembling the standard. The top bar is called the "navigation bar". The navigation bar comes with a standard set of functions such as switching its title according to the view displayed, as well as providing a back button. You can add one or two more buttons, but Apple recommends using only one on each side of the title as to not obscure it. Another button often seen on the navigation bar is a done button. This is usually used for dismissing the keyboard or other controls. You’re allowed to customize the color of the navigation bar to match the overall graphic profile of the app, you can also make it transparent to display graphics and content underneath.
The bottom bar is known as the tab bar. This is used to quickly navigate between sections in the app, and must be visible on all sections. You can add icons to the tabs and make the tab bar slightly transparent, but that's the only customization available. The icons must be gray scale and the blue overlay indicating the active tab is not editable.

There is a clear difference between designing iOS apps and making websites. The CSS (Cascading style sheet) of a website allows practically unlimited customization of any element. Your imagination is usually the only thing that restricts the design. The reason might be that nobody owns the web, and nobody has a right to dictate the look and design of a website. Apple, on the other hand, owns the iOS, and has the right to dictate exactly what is displayed on it, at least software-wise.

But what can I deduce from this that helps my research? Are there any indications already at the development stage of the app that implies the finished product will be appreciated and seen as part of the great iOS? I think a clear sign is the fact that Apple is enforcing its authority at a very early stage. Although it might seem harmless, there is quite a detailed document suggesting how you should design your app before you even start (Apple F). Along with all the limitations placed on the UI components, as discussed earlier, I feel that Apple shapes my app into its own image. If this is their plan, you can't really say it's a bad plan. The fact is that Apple themselves made the first apps that came bundled with the iPhone. They also made the iOS. These are the first apps that people started using and defining the iPhone interface by. This means that in shaping user apps, such as my own, into their own model, the UI is given quite a bit of help becoming good, as it is expected to be. This is a very different philosophy from what the developers of Lunarstorm, a Swedish online community, employed (Skog, 2010). Their approach when introducing a new element on their website was to not advertise it, explain how it worked, or give any guidelines. Instead they wanted to place it in the environment and see how the community put it to use. This was also a very successful approach as Lunarstorm became one of the largest communities world-wide. Apple seems to go the complete other direction, giving strict guidelines and enforcing rules on how components should and should not be used. As seen in an earlier quote, they seem somewhat afraid of compatibility consequences that might arise between the apps and the evolving version of the iOS if developers were allowed to shape components and do whatever they liked. Apple seems to want to direct focus on the functionality, making sure that the app is first and foremost filling some sort of purpose, and that this is being done as according to UI guidelines as possible.

4.2.2. The app submission process

As I was getting closer to being finished, I started looking at the next step. First of all I needed to get my app tested on a device, and in order to do this I had to start learning the app submission process. There's two alternatives for app distribution: ad hoc and app store. Ad Hoc means you are allowed to distribute your app to a fixed number of devices, usually for prototype testing. The app store is what it implies; you submit your app for review and hopefully get it approved, after which it is placed in the app store for anyone to buy or download for free. Most apps are intended for the app store, as the ad hoc solution isn't practical for easy distribution. First of all, the developer needs to download a certificate binding a distribution profile to the app. This distribution profile contains each device serial number allowed to install the app, and must be updated each time another serial number is added. You will then also need to download this profile again and refresh the connection to the app, so it recognizes this new serial and may be installed on the respective
device. So clearly this is not an effective means of large-scale distribution, especially as the allowed number of devices to distribute to start at 100 and then refresh slowly over time.\(^3\)

While I thought making the app would be the challenge, it would turn out to be another obstacle trying to figure all this out. Downloading certificates, profiles, key chains, setting up Xcode for distribution and connecting all the threads took a nice bit of time. I feel that the process may be slightly exaggerated, at least for distributing a prototype. Most of the time it took was spent on learning the process, which wouldn't be required again, but it is still a very complex system before even having started the review process.

Apple has tried to answer all questions before they are asked with a 22 paragraph guideline document (Apple C). What is of particular interest for my study are a few lines of paragraph 10; User Interface:

10.1: Apps must comply with all terms and conditions explained in the Apple iOS Human Interface Guidelines

10.3: Apps that do not use system provided items, such as buttons and icons, correctly and as described in the Apple iOS Human Interface Guidelines may be rejected

10.4: Apps that create alternate desktop/home screen environments or simulate multi-app widget experiences will be rejected

10.5: Apps that alter the functions of standard switches, such as the Volume Up/Down and Ring/Silent switches, will be rejected

10.6: Apple and our customers place a high value on simple, refined, creative, well thought through interfaces. They take more work but are worth it. Apple sets a high bar. If your user interface is complex or less than very good it may be rejected

The first criteria states that the app must comply with the guidelines. So they are actually rules rather than guidelines. And these rules more or less imply that the app has to be good. The next point reinforces the model that discreetly shapes the apps into what Apple wants. 10.4 and 10.5 continue to limit the developer in what he is prohibited from using. The final criteria give Apple the right to decline an app based purely on the looks of it.

Although my app has technically been completed for a while, it is still not ready for submission. There are some placeholders in the informational section that the company hasn't decided what to do with yet, keeping me from submitting it for review. It would have been interesting to see how my first app submission fared, but it is not completely certain I would have gotten any response before this report is due. Although Apple reviews most of the app submissions within a week (according to their own figures), it is not uncommon to hear of people having waited longer. A Google search will return a lot of these topics where confused developers are reading "waiting for review" long over a week.

### 4.3. Evaluating the application

A heuristic evaluation is performed in an uncomplicated way. Basically, the expert (me) sits down with the interface (my app) and tries to find any usability problems with the help of a set of heuristics (the Apple Human Interface guidelines). These problems are noted and given an appreciation as to how severe they are. The heuristics have been described and analyzed earlier in 3.2. I have written up my thoughts on each in the table below.

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Notes</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metaphors</td>
<td>The tab bar has icons reflecting their content. The glossary has a sidebar enabling instant navigation to a selected letter. A loading image is displayed when web content is being loaded.</td>
<td>There is no indicator that there is more text than the window displays. The user has to attempt a scroll gesture to find out.</td>
</tr>
<tr>
<td>Reflect the users</td>
<td>The calculator presents a custom control when a</td>
<td></td>
</tr>
</tbody>
</table>

\(^3\)Information in this paragraph comes from [http://developer.apple.com/](http://developer.apple.com/) and sub-pages. Some information may only be accessible for logged in developers.
<table>
<thead>
<tr>
<th>mental model</th>
<th>field is clicked. A value is selected and the user taps the done button when he is. The new calculations are immediately presented. The information section animates to indicate a move up or down in the page hierarchy Web content is &quot;pinched&quot; to zoom in or out.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing complexity in your software</td>
<td>The table view makes navigation of large amounts of information easy. The design is very simple and straightforward.</td>
</tr>
<tr>
<td>Consistency</td>
<td>Each page looks pretty much the same except for the news screen. The app resembles other apps. No unfamiliar gestures or techniques are required.</td>
</tr>
<tr>
<td>Feedback and communication</td>
<td>Any tap will display a &quot;clicked&quot; animation on the respective element. The loading image indicating work in progress. Calculations are updated automatically. Input is displayed as it is typed. An error message is displayed if internet connection is lost. Web content seems to load even if there is no internet connection before error is displayed.</td>
</tr>
<tr>
<td>Modelessness</td>
<td>When navigating the informational section, one step back in the hierarchy is possible. The tab bar is always visible. The last calculation made is saved as long as the app isn't closed. There are no shortcuts in the table navigation. 10 steps down in the hierarchy requires 10 taps.</td>
</tr>
<tr>
<td>Direct manipulation</td>
<td>Input is displayed as it is typed. All taps on tappable elements are responded to. Calculations are updated real-time. If you click one control when another is open, the first control goes down without the second one coming up, requiring another tap.</td>
</tr>
<tr>
<td>Explicit and implied actions</td>
<td>The table view suggests headlines that respond by opening its content in a new view. The calculator displays buttons and titles for each row of data, suggesting where input can be entered.</td>
</tr>
<tr>
<td>Forgiveness</td>
<td>There are steps back when navigating. Calculations are reversible. Last states of views and data are saved if app is closed. Closing web content requires it to be reloaded, losing its current state.</td>
</tr>
<tr>
<td>Perceived stability</td>
<td>The app looks like other apps. Controls are familiar and work as they usually do.</td>
</tr>
<tr>
<td>WYSIWYG</td>
<td>All values in the calculator can be edited. Content is presented &quot;as is&quot;.</td>
</tr>
<tr>
<td>Aesthetic integrity</td>
<td>Except for the navigation bar and tab bar, the layout seems consistent and simple. No additional types or graphical elements are applied. A simple gradient in the background. Content seems almost a bit too plain / naked.</td>
</tr>
<tr>
<td>User Control</td>
<td>No navigation is performed without user taps. The calculator doesn't update without user input. The calculator doesn't suggest any predefined, often used values.</td>
</tr>
</tbody>
</table>

**Table 2: Heuristic evaluation of the Chimney app**

The problems that were found aren’t that serious in their nature, they’re mostly just personal preferences or annoyances I doubt the app would have been declined in the review, but they might have seen more issues with the interface. It seems my end result pass the Apple criteria quite well, which would have been surprising otherwise considering how much of a hand they have had in the design.
5. Conclusion

My research question was: What makes the iOS such a success seen purely from a software point of view? Making the app, experiencing all the Apple guidelines and rules, inclines me to answer that the iOS and all the apps on it are such successes because Apple seems to know a lot about usability. They enforce this quite heavily on developers and double check it with a strict review process. The program used to develop iOS apps is made by Apple and this is their main tool for making sure that apps are sculpted to fit the Apple profile. With their human interface guidelines matching researched heuristics and seeming very adept at evaluating mobile interfaces, it is no coincidence that they are used more like rules than guidelines.

The iOS also makes full use of the hardware, putting multi-touch and the large screen to full use even in my little beginner’s app. The iOS is also designed with ergonomics in mind. Most navigation and input can be done with one hand, without changing the grip on the device and usually just using your thumb. Designing for any physical situation is something that is researched and studied (Vicente, 2007) and made easy with the iOS approach.

Simple and refined aesthetics are highly valued by Apple. The iOS makes use of images and icons as metaphors, but there is little else in the way of clutter. It touches upon not fixing that which isn’t broke. Usability isn’t about flashy graphics and animations, but functionality and simple thinking. The WIMP (Windows, Icons, Menus, Pointers) principle has been a popular term in the computer industry, defining the accepted and widely used graphical interfaces based on these four points. The WIMP principle can be seen adapted on the iPhone, with windows being what they are, icons playing a much larger roll in the interface, menus being used in innovative ways such as the picker wheel, and pointers being the touch input. So the principles used by early pioneers in the computer industry are recycled and used again to define another great interface.

A lot of success probably comes from the fact that Apple has gone the opposite way to the norm in a couple of regards. The interface was made a whole bunch easier even as the technology took large leaps forward. This is confirmed by research underpinning how important it is to simplify when things get more complicated (Dredge, 2008). Another occurrence in which Apple did the opposite was in restricting the developer freedom. Instead of giving them all the options in the world, they were limited quite heavily, making sure the apps produced were in line with Apples model.

6. Continued research

This study could evolve in a couple of different directions. One track would be further exploring the Apple design guidelines. Where have they come from? Another course would be expanding my evaluation. Having 5-10 more experts sit down and evaluate the interface with the proposed heuristics. This would certainly yield more problems, perhaps some large ones which I failed to find. The fact that I did not find any major problems with the iOS is something one could look at and analyze. A different direction would be taking a step back and looking at other reasons for the iPhone success. Perhaps the iOS is only the smallest part of the reasons why the iPhone became popular.

Further, it would be interesting to explore the concept of "context-awareness" (Vicente, 2007). How will Apple apply these new thoughts to their devices? What will come after the touch screen? Will the next product be iThink, a device that does all the thinking for you and automatically presents you with what you want before you even have to ask for it?

Reflecting on the way we have seen Apple work, and when it comes to technology in general, I think the proverb "To know your future you must know your past" fits well.
7. References


Carlson, Nicholas. Business Insider. *And boy have we patented it*. 2010-03-02. 
[http://www.businessinsider.com/and-boy-have-we-patented-it-2010-3](http://www.businessinsider.com/and-boy-have-we-patented-it-2010-3) (Retrieved 2011-05-13)


Graham, Flora. CNET UK. *The iPhone is the worst phone in the world*. 2009-11-03. 


8. Attachments

8.1. Calculator interface

![Video-Film Calculator](image)

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Film format</td>
<td>35 mm 2-perf</td>
</tr>
<tr>
<td>Film length</td>
<td>1713 m</td>
</tr>
<tr>
<td>Frames</td>
<td>179820</td>
</tr>
<tr>
<td>Duration</td>
<td>1 hr 40 min 0 sec</td>
</tr>
<tr>
<td>Disk space</td>
<td>2.24 tb</td>
</tr>
<tr>
<td>Compression</td>
<td>None</td>
</tr>
<tr>
<td>Frame size</td>
<td>1920x1080</td>
</tr>
<tr>
<td>Frame color depth</td>
<td>16 bpp</td>
</tr>
</tbody>
</table>

Uncompressed: 373.07 mb/s
8.2. News / first screen interface