iOS Developer Starter Kit

Table of Content

HISTORY

In the Beginning

Steve Jobs was a 21-year-old college dropout living with his parents in Los Altos, Calif, where he and two friends, Steve Wozniak and Ronald Wayne, would hang out in the garage. For other trios, this would be the beginning of a rock band; but Jobs, "Woz" and Wayne had other things on their minds. Jobs and Wayne had both worked together at the gaming company, Atari, while Wozniak, 26, had worked for Hewlett-Packard. The three men incorporated Apple Computer on April 1, 1976. While the two Steve's went on to greatness as Apple's revolutionary approach to personal computing bore fruit, Wayne sold his share of the newly created Apple for just $800 just three months after its inception.

Introducing Apple I

The first Apple computer seems positively antique by today's standards. Hand-built by Apple co-founder Steve Wozniak in SteveJobs' parents' garage and first introduced at the Home-brew Computer Club inPalo Alto, Calif. In 1976, the Apple 1 was originally a do-it- yourself kit which didn't even come with a case. Even so, as the first all-in-one microcomputer that, once hooked up to a keyboard and monitor, didn't require extra circuitry to display text, it was a giant step forward over the competition. It came as a kit and sold for $666 (not for satanic reasons, but because Wozniak apparently preferred repeating digits). More than 200 units were sold by the Byte Shop, an early computer store. In 1999, the Apple I sealed its place as the most collectible PC of all time — one lucky tech aficionado scored $50,000 for his original Apple I

Dawn of the Mac

When Steve Jobs introduced the Macintosh at the company's annual shareholders meeting on January 24, 1984, pandemonium apparently ensued. At $2,495, the Macintosh was the first affordable computer to offer a graphical user interface, replacing fusty text-based operating systems with an intuitive layout of folders and icons.

If anything, the Macintosh's world debut was even more noteworthy. The Mac's famous Superbowl ad, directed by Alien auteur Ridley Scott, would become a pop icon in its own right, reappearing more than 20 years later as a political parody wielded by tech-savvy Barack Obama supporters. Much of the message behind the ad — Conformity sucks! Non- conformity rules! Apple is for non-conformists! — has stayed, albeit stripped of its none- too-subtle Orwellian overtones.

Computer Candy

The iMac, introduced in 1998, had two characteristics that quickly made it the best-selling personal computer in America. First, it was a self-contained unit that required minimal setup and even had a handle that made it easy to pull out of the box and move around. The elimination of the tangle of device cords that typically powered and connected the computer and monitor made the iMac attractive to users who didn't know much about computers. (According to Apple, one-third of iMac sales in the machine's first year went to first-time computer buyers.) Second, the iMac was pretty. On the shelves next to the beige computers that dominated the market back then, the teal iMac, with its cute circular mouse and translucent body, stood out. Less than a year after Steve Jobs unveiled the iMac, he introduced a second generation that came in five bright colors that a New York Times reporter wrote "more closely resemble a pack of Life Savers than a new computer line." A week after the candy-colored new iMacs became available, Apple announced its most recent quarterly earnings were three times what they had been the year before.

The Release of Mac OS X

While away from Apple and working for NeXT, yet another computer company he founded, Jobs picked up a few new tricks – one very important trick in particular: the operating system that would become Mac OS X ("ten" not the letter X). Released in March 2001, and touted as virtually crash-proof, OS X was also easy on the eyes because of its "aqua" look and feel, noted for its soft edges, translucent colors and pinstripes. OS X continued to evolve, going through nine different iterations in ensuing years; its stability, speed and ease of use became a major selling point for new Mac users who switched from Windows-driven PCs.

The iPhone

In the aftermath of the iPod's success, pundits began debating whether apple would get into the competitive smart phone market. Speculation continued until January 2007, when the company announced the iPhone. the phone's release on June 29, 2007 prompted enormous lines outside of Apple stores as well as adoring praise from reviewers. (It was named TIME's "Invention of the Year.") The phone-music player-pocket computer sold some 1.4 million units by Sept. 30 of that year. But some early adopters felt like suckers after shelling out $599 for the 8GB iPhone and $499 for the 4GB version, only to learn just two months after the release that the company was slashing the price of the phones by $200.

A year later, the iPhone 3G was released at $199. That seemed to be a value proposition which made sense to everyone concerned: Every model since, including this year's iPhone 5, has started at the same price – as long as you sign a new two-year contract with a mobile phone service provider.

iOS OPERATING SYSTEMS

Version	Build	Release date	Terminal update for			
			iPhone	iPod Touch	iPad	Apple TV
3.1.3	7E+18	February 2, 2010	iPhone (1st generation)	iPod Touch (1st generation)	—	—
4.2.1	8C148	November 22, 2010	iPhone 3G	iPod Touch (2nd generation)	—	—
5.1.1	9B206	May 7, 2012	—	iPod Touch (3rd generation)	iPad (1st generation)	—
6.1.6	10B500	February 21, 2014	iPhone 3GS	iPod Touch (4th generation)	—	—
7.1.2	11D257 11D258	June 30, 2014	iPhone 4	—	—	Apple TV (2nd generation) (Apple TV Software 6.2.1)
8.3	12F69 (Apple TV only)	April 8, 2015	—	—	—	Apple TV (3rd generation/3rd generation Rev-A) (Apple TV Software 7.2)
9.1	13B143	October 21, 2015	N/A	N/A	N/A	N/A
9.2 Beta 3	13C71	November 10, 2015	N/A	N/A	N/A	N/A

APP DEVELOPMENT LIFE-CYCLE -> BUSINESS REQUIREMENT

-> APP STORE SUBMISSION

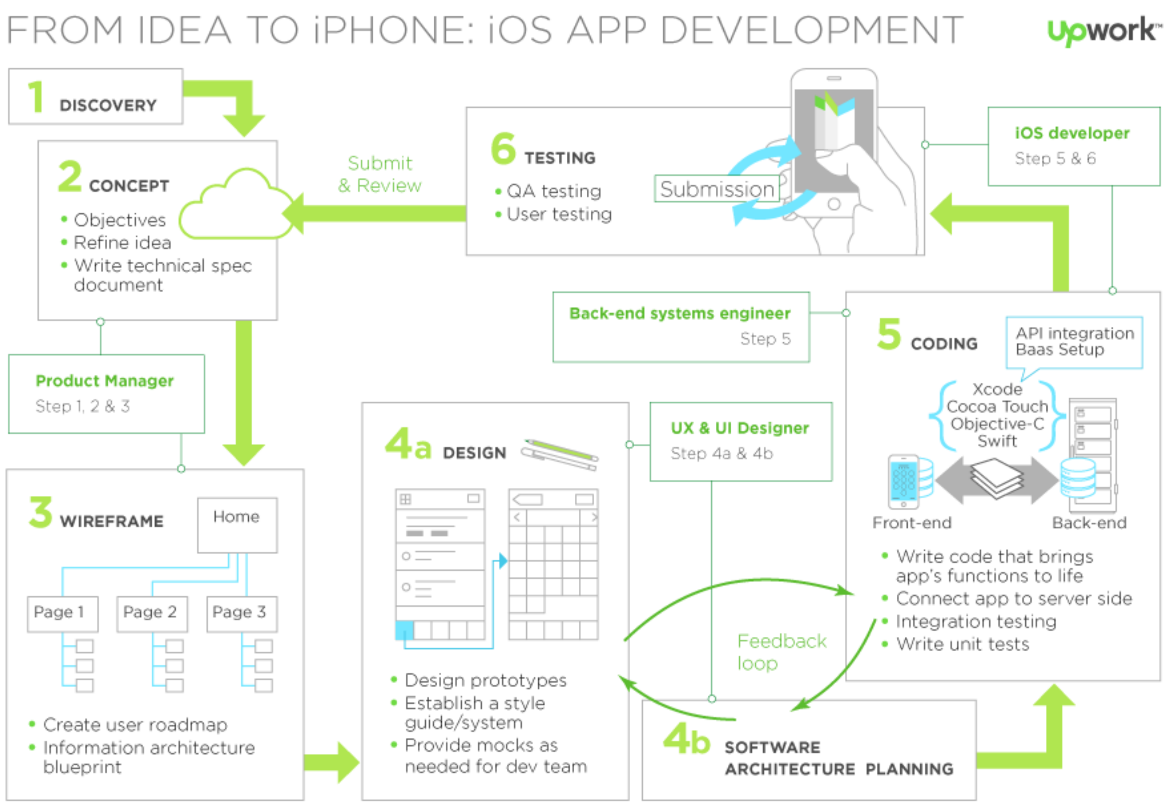
Assuming that you are seeking suggestions on a techno-commercial approach towards your forthcoming App or Game (Let's call it product here), then it should go as follows:

1. Put your idea on paper, along with what you feel is the "core" USP of your product. Do note down what is your maximum expectation out of the product (Number of users only this time, revenues can be gauged later on), and what maximum you've in your mind to be given as features in your product.
2. Make some mock ups (Hand sketched, or professionally designed) of the full product you've just idealised.
3. Divide it into minimum 4 (if you are to attract investors first) or 3 (self funded battle) broad phases, 1st being the working PoC to be shown to potential investors and if that is not pertinent, then make 1st phase as an App with at least the core USP functionality for sure; further 2 or 3 versions should be further features/ functionality or fine-tuning; keep them tentative as user's feedback on first version can change so much in what you'd have noted for further phases.
4. So you've nailed down on phase 1 out of the whole specs; make them finalise and convert them into an SRS as you'd do for any other software development, or web development.
5. Iron out the various components of the Phase 1 development, i.e Software design part, GUI design part, Creatives/ artwork part, actual implementation/ coding part, testing/ QA part, and submission part (tiny though).
6. Estimate the work overall, assign various parts to skilled team according to their expertise, and kick off.
7. If it is to target iOS 5 or later only, then you can quickly create a storyboard of your product first, to finalise the flow of the app; else go with the traditional way of picking up the most core module first and start developing the same. and add further ones subsequently.
8. Keep the QA going on at each interim build (We do weekly interim builds, with continuous QA, ensuring that each interim release is "ship-able". It reduces the probability of spoiling of party at the time of launch.
9. At the final beta, give it maximum number of confidential users to do UAT (and for feedback too), you can pick your family, friends or social circle to do the same for you. Take feedback or bugs reported, and incorporate what you feel is required, leave the trivial ones for further updates but note them down to be included in specifications for

version 2. Get an honest feedback from all please. Also, leave a feedback form on your App and associated website as well, for actual end users to say what they feel! This is most important for any product company, mind it.

10. So lot of fine-tuning and polishing - Now submit it to the store (check that it is compliant to guidelines in the very beginning only please!).
11. Do the promotion/ marketing with the ways most preferred (and seem fruitful) to you.
12. Keep the team working on further most important features (if budget/ time allows) in the mean time for next phase, and also keep collecting feedback and filter those as per your prudence and including the best ones in the ongoing phase 2 development.
13. Keep repeating the process (If success allows, and I wish it would) and keep your product evolved as what user's want, and as what your team think should be the best to offer.

Sorry for being mostly non technical and more towards business side, but I believe that is what is more important. It is a commercial market at the end of the day, and I believe all development approaches should come from the market scenario only, and should be based upon the performance of your product.



OBJECTIVE C vs SWIFT

Observing our favourite Objective C and fresher's favourite Swift, we might not be able to find a big difference. Objective C, which has been in the field since the very beginning of iOS, holds much importance and is quite popular. Simpler Syntax and easy to understand features makes it user-friendly. It is valuable and good but it still lacks some of the features that developers always wanted, just like the Tuple feature introduced in Swift. As per my experience working on both the technologies, it is one of the best feature in Swift. Removal of 'NS' from Arrays, Dictionaries makes Swift provide comprehendible codes and give a much neater look, providing much more readability. Transition from Objective C to Swift

To make the transition from Objective C to Swift a smooth process, Apple made the syntax similar and more familiar. It removed some of the prefixes and added some suffixes to the existing ones. Features like providing the range for iterative statements, switch functionality revamp, type inference, closures, which are similar to blocks, generics, which can take up any data type, are some of the beneficial ones in Swift. And on the applications side, the change that Swift has brought in is the 1 or 2 lines of code in Swift compared to lengthier code in Objective C. Swift supports Unicode characters, as variable names, which Objective C does not.

Apple considers Swift as a safe, modern and powerful coding language. Playground feature, which is present in Swift, allows the user to interact with a running application, side by side, which Objective C cannot. Swift, being a bit high on the graph, claims to provide a better performance.

So, Should you start learning Objective C or Swift?

Almost all current applications are still in Objective C. It will take some years for Swift to gain its place and importance in iOS applications. Yes, it's true that some of the applications are getting converted to Swift, but it would take some time to gain prominence.

While exploring iOS, it is recommended to have good knowledge about Objective C and at the same time, sufficient conceptual knowledge about Swift, too. The ideas are same in both Programming Languages. If you are in the midst of Objective C learning, I would recommend not to start with Swift right away. Gather some good knowledge about Objective C, Then, slowly, get to know about Swift. Learn it and work out on some small projects.

HELPING HAND - Github & stackoverflow.com

HOW TO USE Github

GitHub's a great tool but it's definitely a little confusing the first time around (and, possibly, a few times after that). That's likely why GitHub created software (for OS X and Windows) to make the process a bit easier. Nevertheless, it's good to learn the old-fashioned way otherwise your options in the simplified software won't make sense. Let's start by walking through the basics.

Step One: Sign Up for GitHub

Here comes the easy part: make yourself a GitHub account signing up on the front page. After completing the form, GitHub will sign you in and take you to your empty news feed. In the middle of the page, you'll see the boot camp (pictured to the right). We're going to go through it to set up your account and, later, create your first repository. Click on "Set Up Git" to get started.

Step Two: Install Git

GitHub exists because of a version control application called git. The site is based around how git works, and git is pretty old. It runs via the command line and has no fancy graphical user interface. Since it's made to manage code you wrote, this shouldn't sound too scary. (Of course, as previously mentioned, GitHub did make wonderful software to allow you to use their service without the command line but that won't help you too much unless you know the basics.)

Git works by reading a local code repository (just a folder containing code for your project) on your computer and the mirroring that code elsewhere (in this case, GitHub's servers). Initially we'll commit (i.e. send) your entire local repository to GitHub, but that's just a one-time affair. As you continue to work on your code, you'll simply commit changes. GitHub will then keep track of the changes you made, creating different versions of files so you can revert back to old ones if you want (or just keep track of those changes for other reasons). This is primarily why you'd want to use a version control system like git on your own, but additional benefits surface when using git to manage code with other people working on your project. When multiple developers commit code with git, GitHub becomes a central repository where all the code that everyone's working on can stay in sync. You'll commit your changes, and other developers will pull them (i.e. sync them to their local repository). You'll do the same with their code.

Git makes this all happen, so you need to download the latest version and install it. On OS X, you'll just install the command line app. On Windows, you'll get a few more items. We'll discuss how they work in the next step.

Step Three: Set Up Git

To set up git, you need to make your way into the command line. On OS X, that means launching the Terminal app (Hard Drive -> Applications -> Utilities -> Terminal) and on Windows that means launching the Git Bash app you just installed—not the Windows command prompt. When you're ready, tell git your name like this:

```
git config --global user.name "Your Name Here"
```

For example, mine would look like this because I'm using a test account for this example:

```
git config --global user.name "Adam Dachis"
```

You can put in any name you like, but afterwards you'll need to input your email and that email must be the email you used when signing up for GitHub:

```
git config --global user.email "your_email@youremail.com"
```

If, for whatever reason, you signed up for GitHub with the wrong email address, you'll need to change it.

Now, to avoid always entering your login credentials and generating SSH keys, you'll want to install the credential helper so your passwords are cached. If you're on Windows, download it and install it. If you're on OS X, you'll need to handle this through the Terminal. To start, use this command to download the credential helper:

```
curl -s -O \
```

This will download a tiny little file and shouldn't take too long. When finished, enter the following command to make sure the permissions are correct on the file you just download (and fix them if not):

```
chmod u+x git-credential-osxkeychain
```

Now it's time to install the credential helper into the same folder where you install git. To do so, enter this command:

```
sudo mv git-credential-osxkeychain `dirname \`which git\``
```

You'll be prompted for your administrator password because the above command began with sudo. Sudo is shorthand for "super user do" and is necessary when performing a task that requires root access. The sudo command allows you to become the root user (a user with permission to do pretty much anything) on your operating system for a moment so you can perform this task. You're asked to enter your password to prove you're an administrator on the computer and should be

allowed to do this. Once you've entered your password and the credential helper has been moved, finish up the installation with this command:

```
git config --global credential.helper osxkeychain
```

Now you're all set and can move on to actually using git and GitHub!

Step Four: Create Your First Repository

Now that you've made it this far, you can actually use GitHub! As a first order of business, we're going to create a repository (or "repo" for short). Head on over to GitHub and click the "New Repository" button on the top right of your account page. (Note: If you're still displaying the GitHub bootcamp section, it'll show up underneath it.)

When creating a repository you have a few things to decide including it's name and whether it'll be publicly accessible or not. Choosing a name should be pretty simple because you likely already have a name for your project. If you're just following along for learning purposes, use "Hello- World." Why "Hello-World" and not "Hello World"? Because spaces and special characters will cause problems. Keep it simple and easy to type in the command line. If you want to include a more complex name, you can add it to the optional description field beneath the name field.

If you're creating an open-source project, you want a public repository. If you want to code by yourself or share only with specific people, a private repository will do. Make the choice that works best for you and your project.

When you're all done, you can click the "Create repository" button but you might want to do one other thing first: check the "Initialize this repository with a README" checkbox. Why? All repositories require a README file. Ideally that file would contain a little information about your project, but you might not want to deal with that right now. By initializing the repository with a README, you'll get an empty README file that you can just deal with later. For the purposes of this tutorial, we're going to leave the box unchecked because, in the next section, we're going to create a README file from scratch to practice committing (sending) it to GitHub.

Step Five: Make Your First Commit

When you send files to GitHub, you commit them. To practice, we're going to initialize your local repository and create a README file to commit as practice. Before you start, you need to know where your local code repository is on your computer and how to access it via the command line. In this tutorial, we're going to assume there's a directory called "Hello-World" in your computer's home folder. If you need to create one, just run this command (same for Git Bash on Windows and OS X's terminal):

```
mkdir ~/Hello-World
```

Now change to that directory using the cd (change directory) command:

```
cd ~/Hello-World
```

In case you were wondering, the ~ represents your home directory in Git Bash and Terminal. It's simply shorthand so you don't have to type it all out (which would look more like /Users/ yourusername/). Now that your repository is ready, type this:

```
git init
```

If you already had a repository ready to go, you'd just need to cd to that directory and then run the git init command in there instead. Either way, your local repository is ready to go and you can start committing code. But wait, you don't have anything to commit! Run this command to create a README file:

```
touch README
```

Let's take a break for a second and see what just happened. Go into the home folder on your computer and look at the Hello-World folder (or look at whatever folder you're using for a local repository). You'll notice a README file inside, thanks to the command you just ran. What you won't see is a .git folder, but that's because it's invisible. Git hides it in there, but because you ran the git init command you know it exists. If you're skeptical, just run the ls command in Git Bash/Terminal to display a list of everything in the current directory (which, if you're following along, is your local repository).

So how does git know we want to commit this README file we just created? It doesn't, and you have to tell it. This command will do the trick:

```
git add README
```

If you want to add other files to commit, you'll use the same command but replace README with the name of a different file. Now, run this command to commit it:

```
git commit -m 'first commit'
```

While the other commands were pretty straightforward, the commit command has a little more going on so let's break it down. When you type git, that's just telling the command line that you want to use the git program. When you type commit, you're telling git you want to use the commit command. Everything that follows those two thing count as options. The first, -m, is what's known as a flag. A flag specifies that you want to do something special rather than just run the commit command. In this case, the -m flag means "message" and what follows it is your commit message (in the example, 'first commit'). The message isn't absolutely necessary (although you'll usually

need to provide one), but simply a reference to help you differentiate the various versions of a file (or files) you commit to your repository.

Your first commit should go by in a split second because you haven't actually uploaded anything yet. To get this empty README file to GitHub, you need to push it with a couple of commands. Here's the first:

```
git remote add origin https://github.com/yourusername/Hello-World.git
```

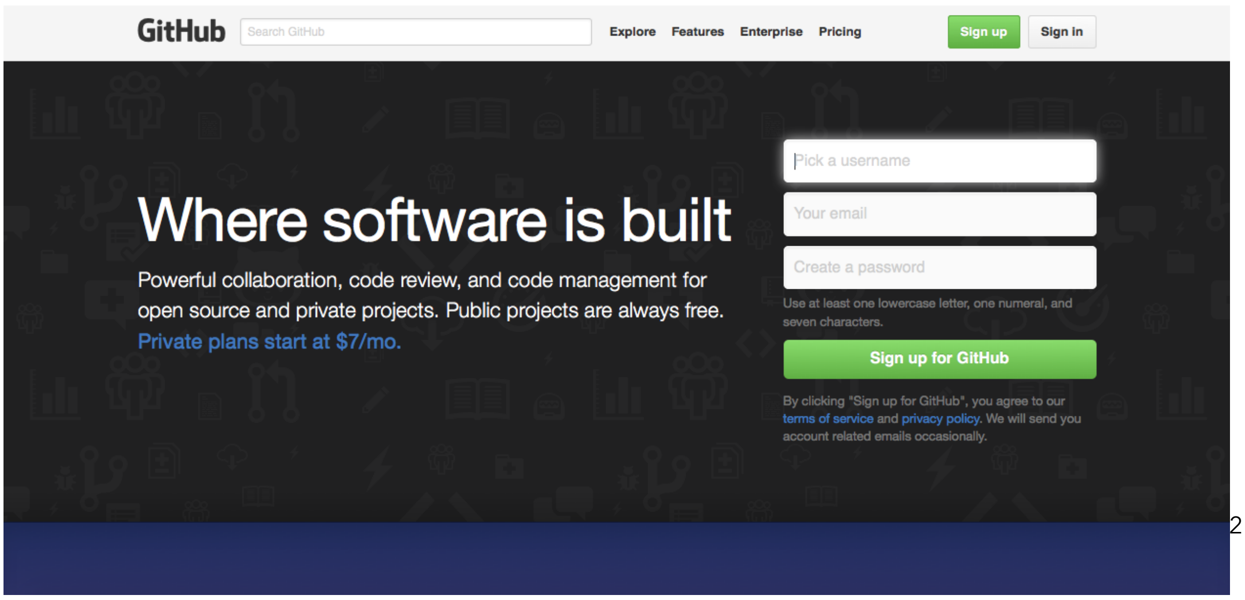
You need to replace "yourusername" with—you guessed it—your GitHub username. For me, it'd look like this:

```
git remote add origin https://github.com/gittest1040/Hello-World.git
```

This command tells git where to send your Hello-World repository. Now all you need to do is send it:

```
git push origin master
```

Once you run that command, everything (in this case, just your README file) will make it's way over to GitHub. Congratulations on your first commit!



Where software is built

Powerful collaboration, code review, and code management for open source and private projects. Public projects are always free. Private plans start at $7/mo.

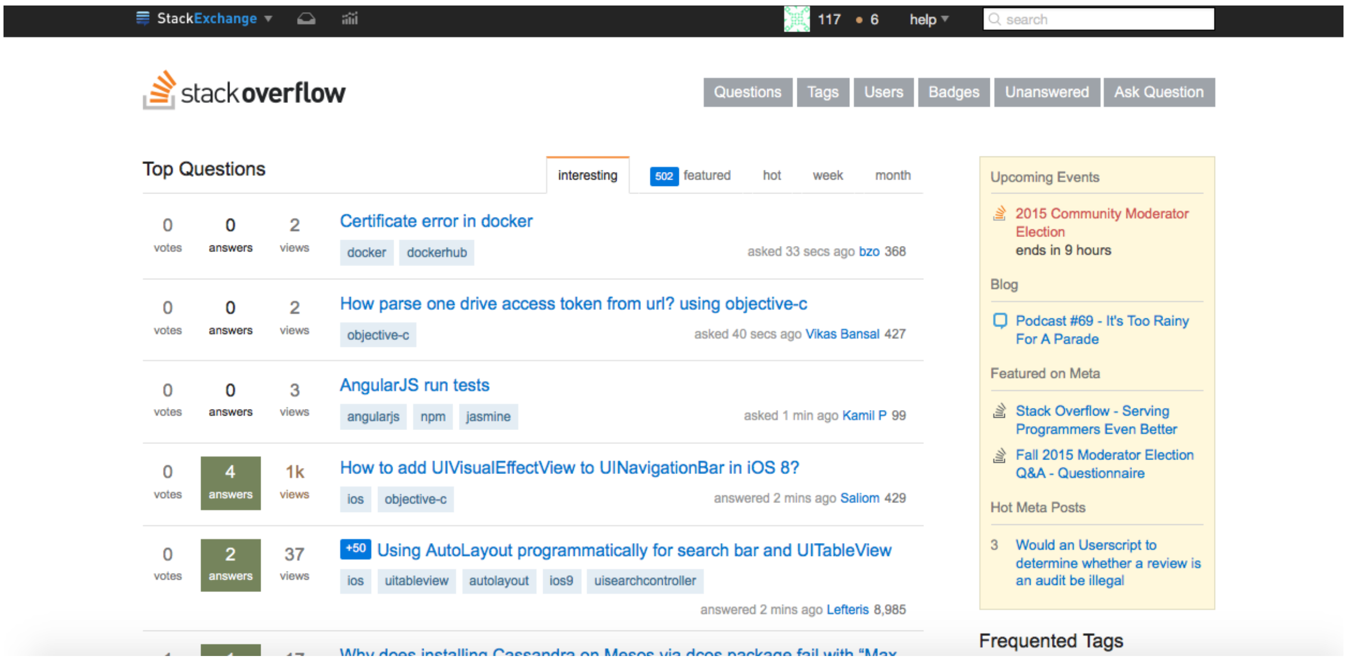
Use at least one lowercase letter, one numeral, and seven characters.

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.

HOW TO USE STACKOVERFLOW.COM

Here's my twelve-step guide to using SO

1. Turn on the computer
2. Wait for it to load
3. Log in
4. Wait for it to load
5. Open up Chrome
6. Click the "Stack Overflow" thumbnail
7. Look at the list of unanswered questions
8. Click a few of them with the middle mouse button if they seem interesting
9. Read them in order. Some will suck, some will be decent; occasionally one has learnable information in it: this is the information that is to be learned!
10. Downvote things that suck, upvote things that rock
11. Complain about the sucky ones with the guys on Google Wave
12. Write a poem to Bill the Lizard in a flag for moderator attention



Top Questions

interesting | 502 featured | hot | week | month

0 votes | 0 answers | 2 views
Certificate error in docker
docker dockerhub
asked 33 secs ago bzo 368

0 votes | 0 answers | 2 views
How parse one drive access token from url? using objective-c
objective-c
asked 40 secs ago Vikas Bansal 427

0 votes | 0 answers | 3 views
AngularJS run tests
angularjs npm jasmine
asked 1 min ago Kamil P 99

0 votes | 4 answers | 1k views
How to add UIVisualEffectView to UINavigationBar in iOS 8?
ios objective-c
answered 2 mins ago Saliom 429

0 votes | 2 answers | 37 views
+50 Using AutoLayout programmatically for search bar and UITableView
ios uitableview autolayout ios9 uisearchcontroller
answered 2 mins ago Lefteris 8,985

Why does installing Cassandra on Mesos via dcos package fail with "Max

Upcoming Events
2015 Community Moderator Election
ends in 9 hours

Blog
Podcast #69 - It's Too Rainy For A Parade

Featured on Meta
Stack Overflow - Serving Programmers Even Better
Fall 2015 Moderator Election Q&A - Questionnaire

Hot Meta Posts
3 Would an Userscript to determine whether a review is an audit be illegal

Frequented Tags

CHEAT SHEET

Objective-C Cheat sheet & Quick Reference

Class Header (.h)

```
#import "AnyHeaderFile.h" @interface ClassName : SuperClass

// define public properties // define public methods

@end
```

Class Implementation (.m)

```
#import "YourClassName.h"

@interface ClassName ()
// define private properties // define private methods @end

@implementation ClassName {
// define private instance variables }

// implement methods

@end
```

Defining Methods

```
- (type)doIt;
- (type)doItWithA:(type)a; - (type)doItWithA:(type)a

b:(type)b;
```

Implementing Methods

```
- (type)doItWithA:(type)a b:(type)b {

// Do something with a and b...

return retVal;

}
```

Creating an Object

```
ClassName * myObject = [[ClassName alloc] init];
```

Calling a Method

```
[myObject doIt];
[myObject doItWithA:a]; [myObject doItWithA:a b:b];
```

Using Properties

```
[myObject setPropertyName:a]; myObject.propertyName = a; // alt

a = [myObject propertyName];
```

```
a = myObject.propertyName; // alt
```

What is a Property?

1) Automatically defines a private instance variable:

```
type _propertyName;
```

2) Automatically creates a getter and setter:

```
- (type)propertyName;
- (void)setPropertyName:(type)name;
```

Using `_propertyName` uses the private instance variable directly. Using `self.propertyName` uses the getter/setter.

Custom Initializer Example

```
- (id)initWithParam:(type)param { if ((self = [super init])) {

_propertyName = param; }

                   return self;
              }
```

NSString Quick Examples

```
NSString *personOne = @"Ray"; NSString *personTwo = @"Shawn"; NSString *combinedString =

[NSString stringWithFormat: @"%@: Hello, %@!", personOne, personTwo];

NSLog(@"%@", combinedString);
NSString *tipString = @"24.99";
float tipFloat = [tipString floatValue];
```

NSArray Quick Examples

```
NSMutableArray *array =
[@[person1, person2] mutableCopy];

[array addObject:@"Waldo"]; NSLog(@"%d items!", [array count]); for (NSString *person in
array) {

NSLog(@"Person: %@", person); }

NSString *waldo = array[2];
```

User eXperience Design

Points vs Pixels

Most Apple docs will use points (pts) as the unit of measure instead of pixels (px). What is the difference? Pixels represent a single dot on a screen so a 5x5 image is made up of 25 pixels (5 wide, 5 high). Point is a unit of length that is independent of resolution. How many pixels make up a point is dependent on the resolution of the screen.

Table 1

Device	**Portrait (px)**	**Landscape (px)**	**PPI**
iPhone 6 Plus	1242 x 2208 (downsample to 1080 x 1920)	2208 x 1242 (downsample to 1920 x 1080)	401
iPhone 6	750 x 1334	1334 x 750	326
iPhone 5 iPhone 5, 5S, 5C	640 x 1136	1136 x 640	326
iPhone 4/4S	640 x 960	960 x 640	326
iPhone & iPod Touch 1st, 2nd, and 3rd Generation	320 x 480	480 x 320	163
Retina iPad iPad 3, iPad 4, iPad Air, iPad Air 2	1536 x 2048	2048 x 1536	264
iPad Mini Retina iPad Mini 2, iPad Mini 3	1536 x 2048	2048 x 1536	326
iPad Mini	768 x 1024	1024 x 768	163
iPad 1st and 2nd Generation	768 x 1024	1024 x 768	132

So the navbar on an iPhone 3 can appear to be the same height as the nav bar on an iPhone 5. It doesn't matter that they have different resolutions (pixels), they share the same common height in points.

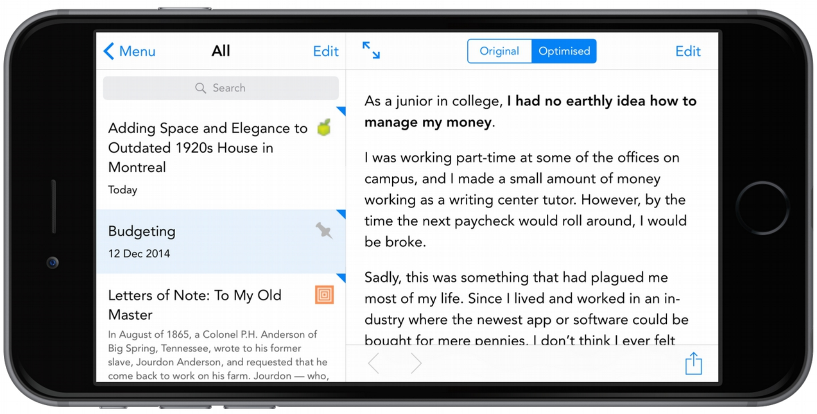
Resolutions

Display

If you are using Adobe Photoshop or another graphics program that lets you adjust ppi of your images then make sure they are set to 72 ppi for consistency since you have no control over the ppi of the device.

Device	App Icon (px)	AppStore Icon (px)	Spotlight Icon (px)	Settings Icon (px)
iPhone 5 iPhone 5, 5S, 5C	120 x 120	1024 x 1024	80 x 80	58 x 58
iPhone 4/4S	120 x 120	1024 x 1024	80 x 80	58 x 58
Retina iPad iPad 3, iPad 4, iPad Air	152 x 152	1024 x 1024	80 x 80	58 x 58
iPad Mini Retina iPad Mini 2, iPad Mini 3	152 x 152	1024 x 1024	80 x 80	58 x 58
iPad Mini	76 x 76	512 x 512	40 x 40	29 x 29
iPad 1st and 2nd Generation	76 x 76	512 x 512	40 x 40	29 x 29

Landscape



Adding Space and Elegance to Outdated 1920s House in Montreal
Today

Budgeting
12 Dec 2014

Letters of Note: To My Old Master
In August of 1865, a Colonel P.H. Anderson of Big Spring, Tennessee, wrote to his former slave, Jourdon Anderson, and requested that he come back to work on his farm. Jourdon — who,

As a junior in college, **I had no earthly idea how to manage my money**.

I was working part-time at some of the offices on campus, and I made a small amount of money working as a writing center tutor. However, by the time the next paycheck would roll around, I would be broke.

Sadly, this was something that had plagued me most of my life. Since I lived and worked in an industry where the newest app or software could be bought for mere pennies. I don't think I ever felt

Portrait

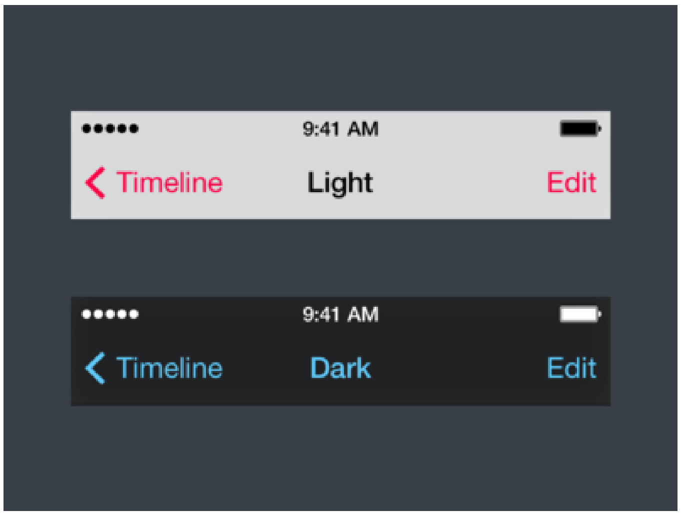
Icons & Images

Dimensions

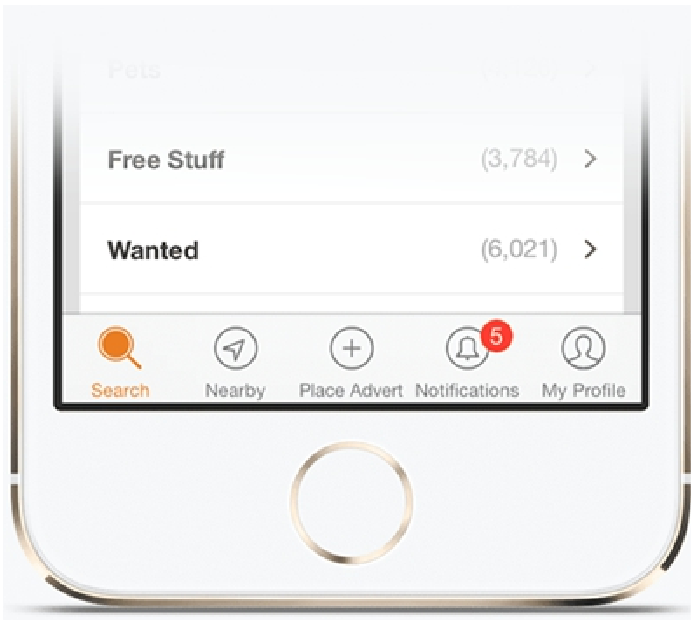
Device	Launch Image (px)	Toolbar and Nav Bar Icon (px)	Tab Bar Icon (px)	Web Clip Icon (px)
iPhone 5 iPhone 5, 5S, 5C	640 x 1136	44 x 44	50 x 50 (max: 96 x 64)	120 x 120
iPhone 4/4S	640 x 960	44 x 44	50 x 50 (max: 96 x 64)	120 x 120
Retina iPad iPad 3, iPad 4, iPad Air	1536 x 2048 (portrait) 2048 x 1536 (landscape)	44 x 44	850 x 50 (max: 96 x 64)	152 x 152
iPad Mini Retina iPad Mini 2, iPad Mini 3	1536 x 2048 (portrait) 2048 x 1536 (landscape)	44 x 44	50 x 50 (max: 96 x 64)	152 x 152
iPad Mini	768 x 1024 (portrait) 1024 x 768 (landscape)	22 x 22	25 x 25 (max: 48 x 32)	76 x 76

iPad 2nd Generation	768 x 1024 (portrait) 1024 x 768 (landscape)	22 x 22	25 x 25 (max: 48 x 32)	76 x 76

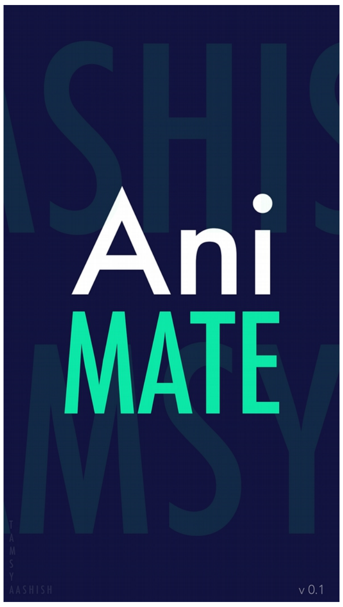
Status Bar & Navigation Bar



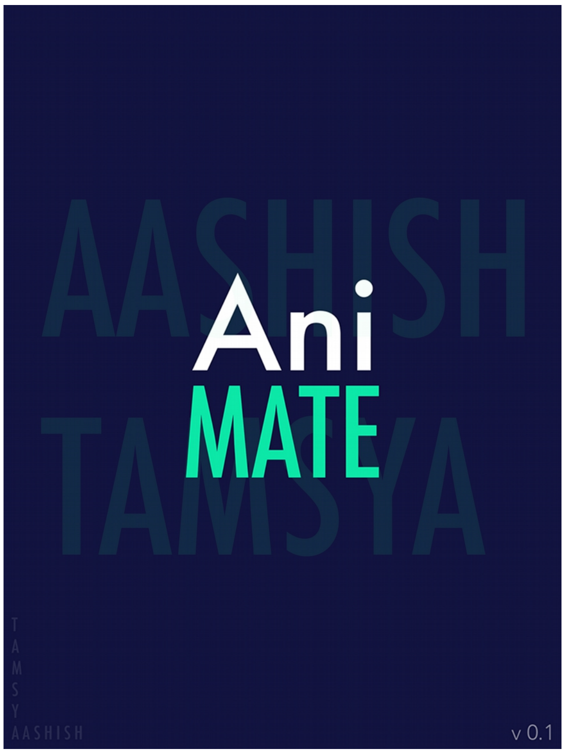
Tab Bar



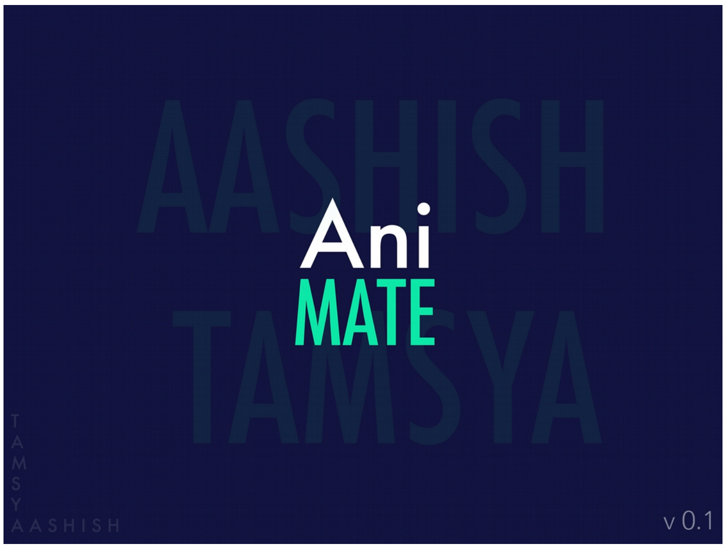
Launch Image



iPhone Portrait



iPad Portrait



iPad Landscape

Device	App Icon (px)	AppStore Icon (px)	Spotlight Icon (px)	Settings Icon (px)
iPhone 5 iPhone 5, 5S, 5C	120 x 120	1024 x 1024	80 x 80	58 x 58
iPhone 4/4S	120 x 120	1024 x 1024	80 x 80	58 x 58
Retina iPad iPad 3, iPad 4, iPad Air	152 x 152	1024 x 1024	80 x 80	58 x 58
iPad Mini Retina iPad Mini 2, iPad Mini 3	152 x 152	1024 x 1024	80 x 80	58 x 58
iPad Mini	76 x 76	512 x 512	40 x 40	29 x 29
iPad 1st and 2nd Generation	76 x 76	512 x 512	40 x 40	29 x 29

For all images and icons, Apple recommends that you use PNG. Avoid using interlaced PNGs.

For icons you need to make sure your icon has 90 degree corners because Apple automatically applies a mask to give it the rounded corner look.

If you wish to create icons that look similar to those used in iOS, use a 2px stroke for detailed icons and a 3px stroke for less detailed ones.

UI

Commonly Used Design Elements

Device	Status Bar Height (px)	Navigation Bar Height (px)	Tab Bar Height (px)	Table Width (px)
iPhone 5 iPhone 5, 5S, 5C	40	88	98	640 / 1136
iPhone 4/4S	40	88	98	640 / 960
Retina iPad iPad 3, iPad 4, iPad Air	40	88	112	dynamic
iPad Mini Retina iPad Mini 2, iPad Mini 3	40	88	112	dynamic
iPad Mini	20	44	56	dynamic
iPad 1st and 2nd Generation	20	44	56	dynamic

HOW TO WRITE CLEAN & OPTIMIZED CODE

Coding is a craft and can therefore be practiced, honed, and refined. Some general tips for developing a "clean" programming style:

1. **Architecture** - draw one to understand where your component or service fits
2. **Workflow** - create one that illustrates the entry and exit points into your code
3. **Pseudo-code** - write out your algorithm(s) before you get in front of an IDE
4. **Design** – have technical leads review your design to vet any key decisions
5. **Documentation** – draft your release notes first to focus on the end-user
6. **Code reviews** – ask for them but also participate as a reviewer
7. **Testing** – keep in mind that a QAE or script will validate and verify
8. **Debugging** – give yourself tools to help during testing and post-release
9. **Exception Handling** – plan for errors and think beyond the "happy path"
10. **Corner Cases** – look at load & edge scenarios that appear well after DAy 1
11. **Metrics** – define and incorporate them so you can improve things in v2.0
12. **Readability** – use a clear convention for variables, functions, classes, etc.
13. **Profiling** – use tools to understand your 80/20 code execution path
14. **Maintenance** – your code will outlast your tenure - think of the next guy
15. **Spacing** – this is minor but clear begin/end syntax is easier to follow

AGILE SOFTWARE DEVELOPMENT METHODOLOGY

Brief overview of Agile Methodology

- In traditional software development methodologies like Waterfall model, a project can take several months or years to complete and the customer may not get to see the end product until the completion of the project.
- At a high level, non-Agile projects allocate extensive periods of time for Requirements gathering, design, development, testing and UAT, before finally deploying the project.
- In contrast to this, Agile projects have Sprints or iterations which are shorter in duration (Sprints/iterations can vary from 2 weeks to 2 months) during which pre-determined features are developed and delivered.
- Agile projects can have one or more iterations and deliver the complete product at the end of the final iteration.

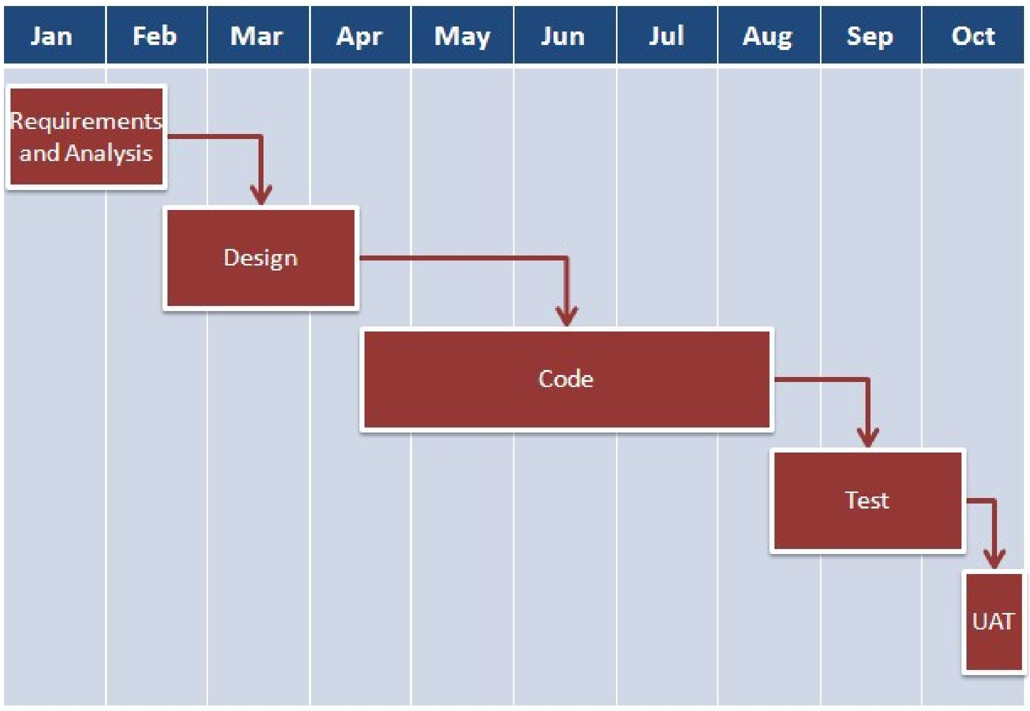
Example of Agile software development

Google is working on project to come up with a competing product for MS Word, that provides all the features provided by MS Word and any other features requested by the marketing team. The final product needs to be ready in 10 months of time. Let us see how this project is executed in traditional and Agile methodologies.

In traditional Waterfall model –

- At a high level, the project teams would spend 15% of their time on gathering requirements and analysis (1.5 months)
- 20% of their time on design (2 months)
- 40% on coding (4 months) and unit testing
- 20% on System and Integration testing (2 months).
- At the end of this cycle, the project may also have 2 weeks of User Acceptance testing by marketing teams.
- In this approach, the customer does not get to see the end product until the end of the project, when it becomes too late to make significant changes.

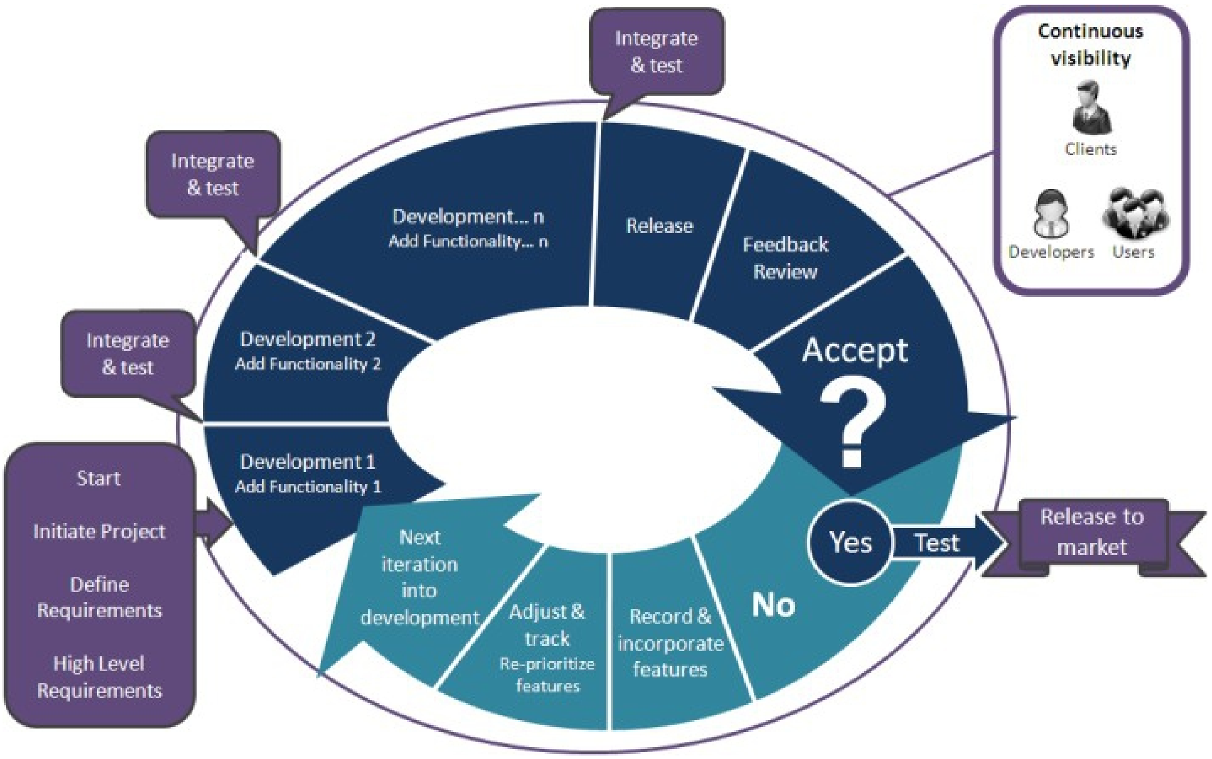
The image below shows how these activities align with the project schedule in traditional software development.

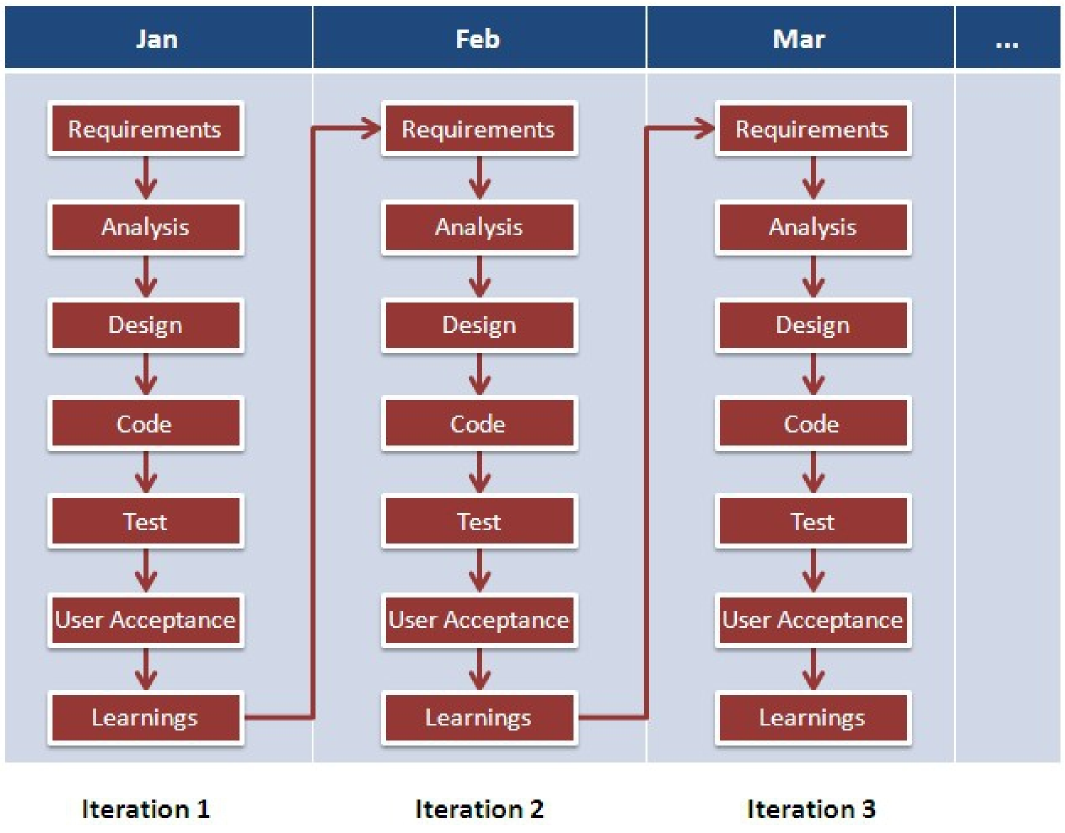


With **Agile development** methodology –

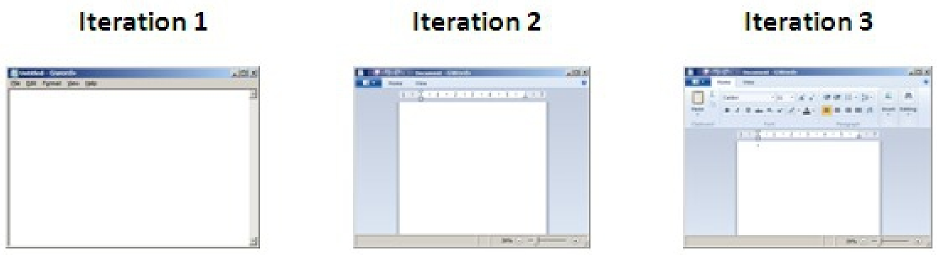
- In the Agile approach, each project is broken up into several 'Iterations'.
- All Iterations should be of the same time duration (between 2 to 8 weeks).
- At the end of each iteration, a working product should be delivered.
- In simple terms, in the Agile approach the project will be broken up into 10 releases (assuming each iteration is set to last 4 weeks).
- Rather than spending 1.5 months on requirements gathering, in Agile software development, the team will decide the basic core features that are required in the product and decide which of these features can be developed in the first iteration.
- Any remaining features that cannot be delivered in the first iteration will be taken up in the next iteration or subsequent iterations, based on priority.
- At the end of the first iterations, the team will deliver a working software with the features that were finalized for that iteration.
- There will be 10 iterations and at the end of each iteration the customer is delivered a working software that is incrementally enhanced and updated with the features that were shortlisted for that iteration.

The iteration cycle of an Agile project is shown in the image below.





This approach allows the customer to interact and work with functioning software at the end of each iteration and provide feedback on it. This approach allows teams to take up changes more easily and make course corrections if needed. In the Agile approach, software is developed and released incrementally in the iterations. An example of how software may evolve through iterations is shown in the image below.



Agile methodology gives more importance to collaboration within the team, collaboration with the customer, responding to change and delivering working software.

Agile development has become common place in IT industry. In a recent survey over 52% of respondents said that their company practiced Agile development in one form or another. Irrespective of your role in the organization, it has become essential to understand how Agile development works and how it differs from other forms of software development.

In traditional approach each job function does its job and hands over to the next job function. The previous job functions have to signoff before it is handed over the next job function authenticating that the job is full and complete in all aspects. For example, Requirement gathering is completed and handed over to design phase and it is subsequently handed over to development and later to testing and rework. Each job function is a phase by itself.

In Agile way of working, each feature is completed in terms of design, development, code, testing and rework, before the feature is called done. There are no separate phases and all the work is done in single phase only.

Advantages of Agile Methodology

- In Agile methodology the delivery of software is unremitting.
- The customers are satisfied because after every Sprint working feature of the software is delivered to them.
- Customers can have a look of the working feature which fulfilled their expectations.
- If the customers has any feedback or any change in the feature then it can be accommodated in the current release of the product.
- In Agile methodology the daily interactions are required between the business people and the developers.
- In this methodology attention is paid to the good design of the product.
- Changes in the requirements are accepted even in the later stages of the development.

Disadvantages of the Agile Methodology

- In Agile methodology the documentation is less.
- Sometimes in Agile methodology the requirement is not very clear hence it's difficult to predict the expected result.
- In few of the projects at the starting of the software development life cycle it's difficult to estimate the actual effort required.
- The projects following the Agile methodology may have to face some unknown risks which can affect the development of the project.

IMPORTANT INTERVIEW QUESTIONS

iOS interview Questions for Freshers

***Q: How would you create your own custom view?**

A:By Subclassing the UIView class.

***Q: What is App Bundle?**

A:When you build your iOS app, Xcode packages it as a bundle. A **bundle** is a directory in the file system that groups related resources together in one place. An iOS app bundle contains the app executable file and supporting resource files such as app icons, image files, and localized content.

***Q: Whats fast enumeration?**

A:Fast enumeration is a language feature that allows you to enumerate over the contents of a collection. (Your code will also run faster because the internal implementation reduces message send overhead and increases pipelining potential.)

***Q: Whats a struct?**

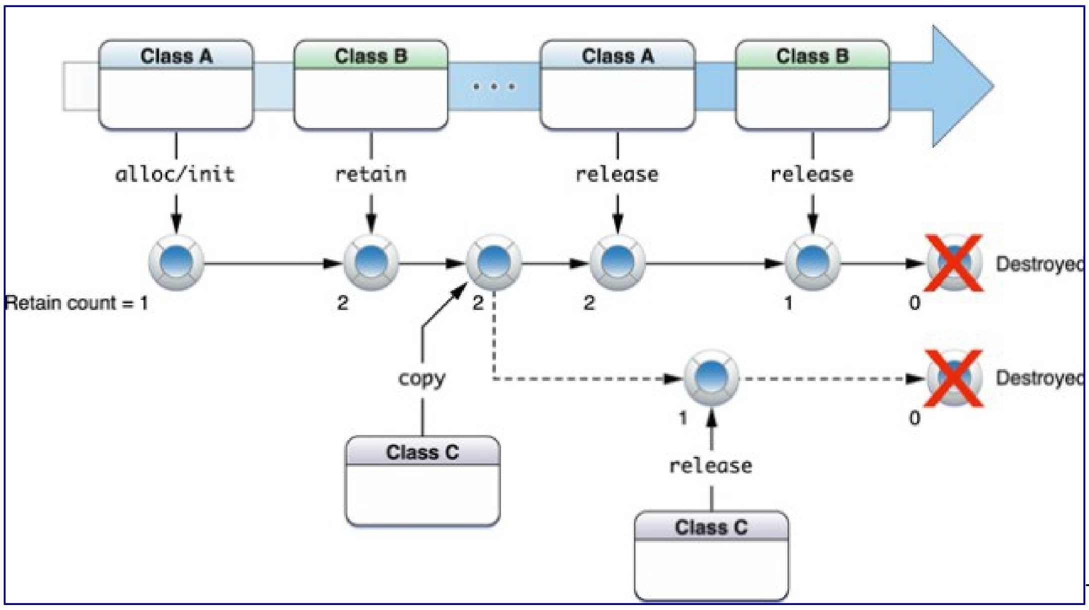
A:A struct is a special C data type that encapsulates other pieces of data into a single cohesive unit. Like an object, but built into C.

***Q: Whats the difference between NSArray and NSMutableArray?**

A:NSArray's contents can not be modified once it's been created whereas a NSMutableArray can be modified as needed, i.e items can be added/removed from it.

***Q: Explain retain counts.**

A:Retain counts are the way in which memory is managed in Objective-C. When you create an object, it has a retain count of 1. When you send an object a retain message, its retain count is incremented by 1. When you send an object a release message, its retain count is decremented by 1. When you send an object a autorelease message, its retain count is decremented by 1 at some stage in the future. If an object's retain count is reduced to 0, it is deallocated.



This will explain how the memory management is done in iOS

***Q: Whats the difference between frame and bounds?**

A:The frame of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to the superview it is contained within. The bounds of a view is the rectangle, expressed as a location (x,y) and size (width,height) relative to its own coordinate system (0,0).

***Q: Is a delegate retained?**

A:No, the delegate is never retained! Ever!

***Q:Outline the class hierarchy for a UIButton until NSObject.**

A:UIButton inherits from UIControl, UIControl inherits from UIView, UIView inherits from UIResponder, UIResponder inherits from the root class NSObject.

*Q:

What are the App states. Explain them?

A:

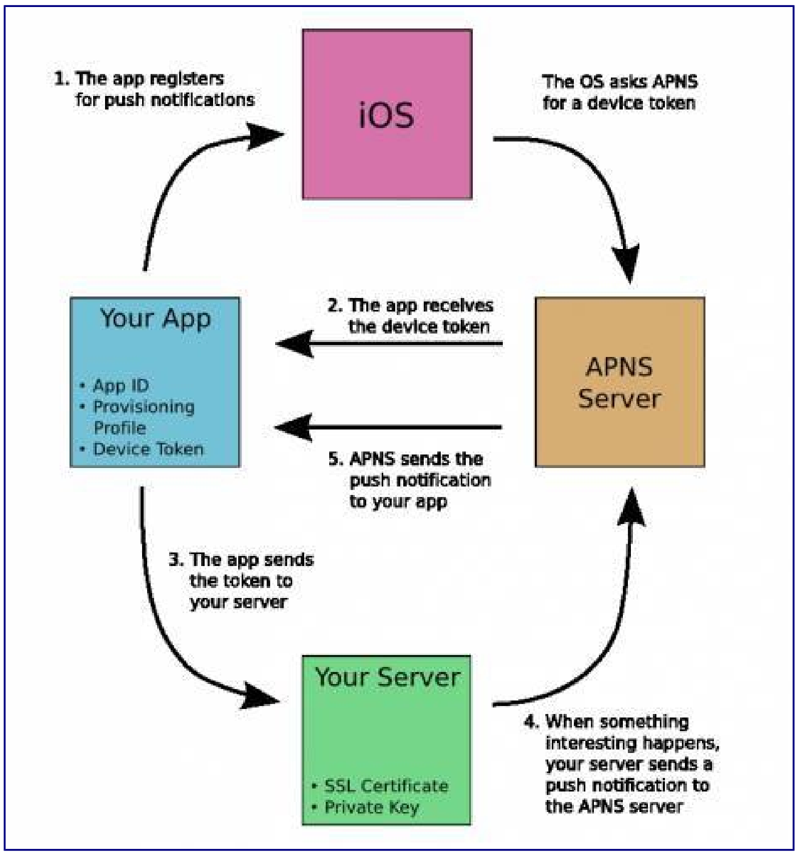
- **Not running State:** The app has not been launched or was running but was terminated by the system.
- **Inactive state:** The app is running in the foreground but is currently not receiving events. (It may be executing other code though.) An app usually stays in this state only briefly as it transitions to a different state. The only time it stays inactive for any period of time is when the user locks the screen or the system prompts the user to respond to some event, such as an incoming phone call or SMS message.
- **Active state:** The app is running in the foreground and is receiving events. This is the

normal mode for foreground apps.

- **Background state:** The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended. However, an app that requests extra execution time may remain in this state for a period of time. In addition, an app being launched directly into the background enters this state instead of the inactive state. For information about how to execute code while in the background, see "Background Execution and Multitasking."
- **Suspended state**:The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute any code. When a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app.

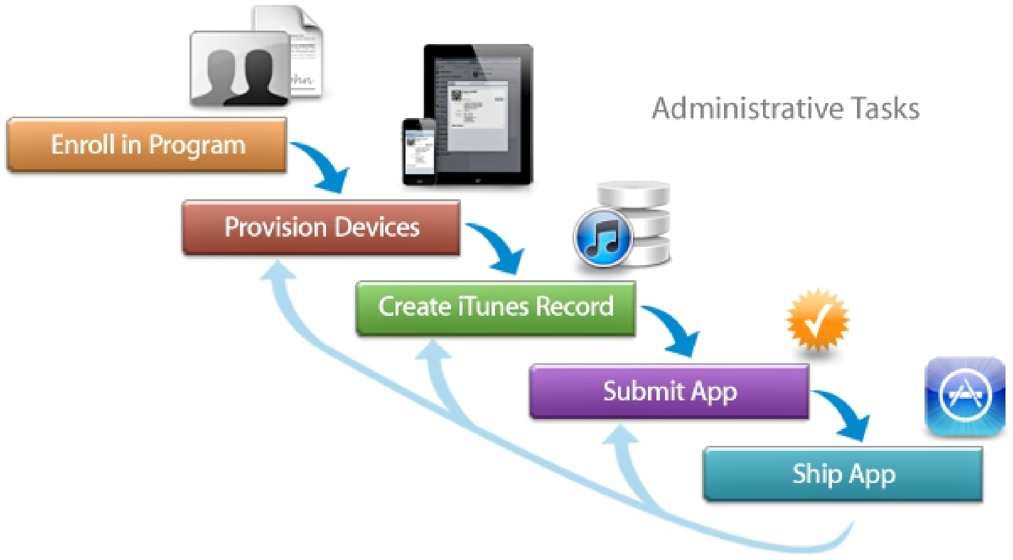
***Q: Explain how the push notification works.**

A:


1. The app registers for push notifications
iOS
The OS asks APNS for a device token
Your App
• App ID
• Provisioning Profile
• Device Token
2. The app receives the device token
APNS Server
5. APNS sends the push notification to your app
3. The app sends the token to your server
Your Server
• SSL Certificate
• Private Key
4. When something interesting happens, your server sends a push notification to the APNS server

***Q: Explain the steps involved in submitting the App to App-Store.**

A:



Apple provides the tools you need to develop, test, and submit your iOS app to the App Store. To run an app on a device, the device needs to be provisioned for development, and later provisioned for testing. You also need to provide information about your app that the App Store displays to customers and upload screenshots. Then you submit the app to Apple for approval. After the app is approved, you set a date the app should appear in the App Store as well as its price. Finally, you use Apple's tools to monitor the sales of the app, customer reviews, and crash reports. Then you repeat the entire process again to submit updates to your app.

Ref: [App Store Review Guidelines]()

***Q: Why do we need to use @Synthesize?**

A:

We can use generated code like nonatomic, atmoic, retain without writing any lines of code. We also have getter and setter methods. To use this, you have 2 other ways: @synthesize or @dynamic: @synthesize, compiler will generate the getter and setter automatically for you, @dynamic: you have to write them yourself.@property is really good for memory management, for example: retain.How can you do retain without @property?

```
if (_variable != object)
{
      [_variable release];
      _variable = nil;
      _variable = [object retain];
}
```

How can you use it with @property?self.variable = object; When we are calling the above line, we actually call the setter like [self setVariable:object] and then the generated setter will do its job.

***Q: Multitasking support is available from which version?**

A:

iOS 4.0.

***Q: How many bytes we can send to apple push notification server?**

A:

256bytes.

***Q: What is code signing?**

A:

Signing an application allows the system to identify who signed the application and to verify that the application has not been modified since it was signed. Signing is a requirement for submitting to the App Store (both for iOS and Mac apps). OS X and iOS verify the signature of applications downloaded from the App Store to ensure that they they do not run applications with invalid signatures. This lets users trust that the application was signed by an Apple source and hasn't been modified since it was signed.

Xcode uses your digital identity to sign your application during the build process. This digital identity consists of a public-private key pair and a certificate. The private key is used by cryptographic functions to generate the signature. The certificate is issued by Apple; it contains the public key and identifies you as the owner of the key pair.

In order to sign applications, you must have both parts of your digital identity installed. Use Xcode or Keychain Access to manage your digital identities. Depending on your role in your development team, you may have multiple digital identities for use in different contexts. For example, the identity you use for signing during development is different from the identity you user for distribution on the App Store. Different digital identities are also used for development on OS X and on iOS.

An application's executable code is protected by its signature because the signature becomes invalid if any of the executable code in the application bundle changes. Resources such as images and nib files are not signed; a change to these files does not invalidate the signature.

An application's signature can be removed, and the application can be re-signed using another digital identity. For example, Apple re-signs all applications sold on the App Store. Also, a fully-tested development build of your application can be re-signed for submission to the App Store. Thus the signature is best understood not as indelible proof of the application's origins but as a verifiable mark placed by the signer