Startup Engineering
Balaji S. Srinivasan
Stanford University

Class: class.coursera.org/startup-001
Signup: coursera.org/course/startup
Forums: bit.ly/startup-eng-forum

Mobile

The Age of Internetification

The basic assumption behind mobile computing is that we are living through an age of internetification, similar to the age of electrification in the early 20th century. Over the span of several decades, electricity evolved from a World's Fair curiosity, to a sensation in which electrical engineers were the computer scientists of their day, and ultimately to a utility whose ubiquity and simple UI belies the stunning complexity of the electrical grid. We don't think about it much today, but our world became *electrical*: every device that could conceivably benefit from electrical power became powered by electricity. Edison's electric light bulb goosed the buildout of the electrical grid, and it was soon used to power irons, sewing machines, and (over time) factories.

In the same way, today we are seeing internet connectivity evolve from flaky 56k modems towards the idea of wireless broadband as a *utility* whose presence you can increasingly simply assume. Just like the process of electrification packaged up an enormously complicated grid into the trivial UI of a wall plug, we've likewise started to move from wrestling with Trumpet WinSock configuration to the trivial UI of an Airplane Mode switch. And with internetification our world will become *mobile*: every device that can conceivably benefit from an internet connection will be connected to the internet. Jobs' internet-connected smartphone goosed the buildout of the mobile internet, but it is now being tasked with connecting locks and thermometers and mines.

The long-term logic of internetification is inescapable, which is why mobile computing is not just a *trend*, but the future of computing (Figures 1, 2, 3). Mobile web applications in particular promise to leave behind desktop web applications in the same way that desktop web apps left behind native desktop apps, and for similar reasons. The fundamental user and developer conveniences afforded by an application downloaded from the internet and continuously updated (i.e. a webapp) generally trumped the fancier widgets that desktop apps provided. Similarly, the sheer convenience of having an app that is constantly in your hands at all times - and that can either change its state based on your location or else make your physical location entirely irrelevant - will increasingly trump the fancier widgets and increased screen real estate afforded by desktop web apps (1, 2, 3, 4, 5).

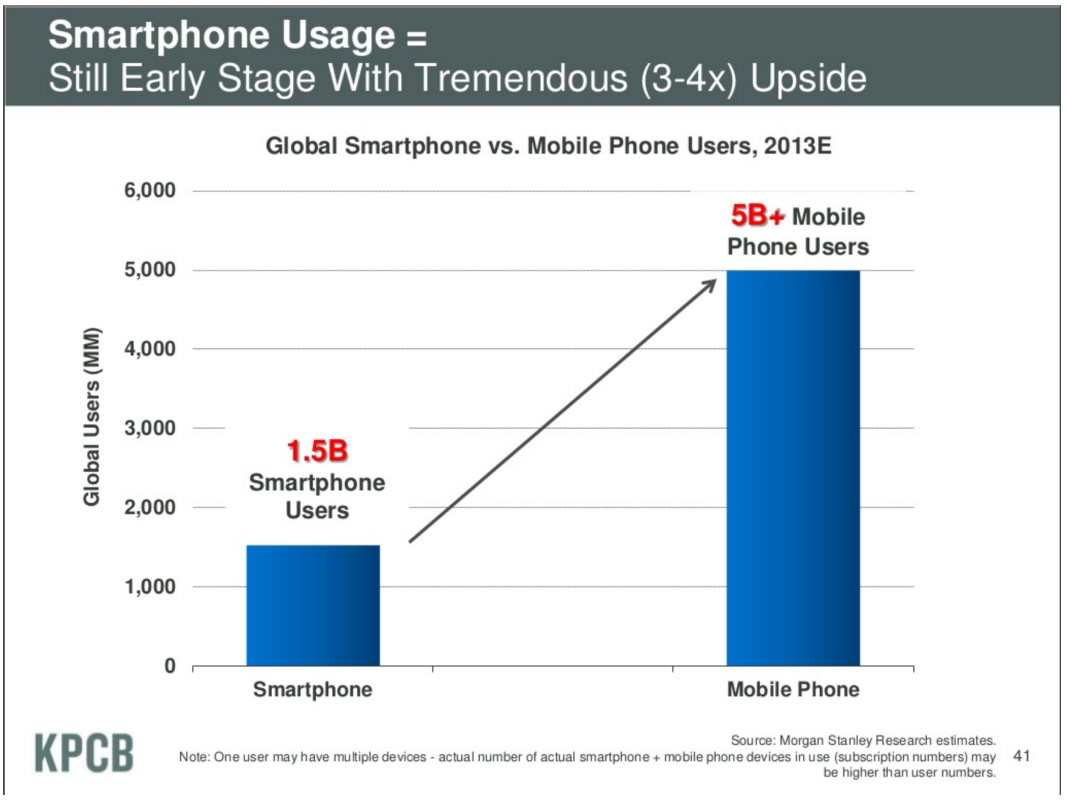


Figure 1: *With an estimated world population of 7e9, global mobile penetration is roughly 70% for all phones and 20% for smartphones (Source: Internet Trends by Mary Meeker at KPCB, who attributes the statistic to Morgan Stanley Equity Research).*

Smartphone Subscriber Growth = Remains Rapid
1.5B Subscribers, 31% Growth, 21% Penetration in 2013E

Rank	Country	2013E Smartphone Subs (MM)	Smartphone as % of Total Subs	Smartphone Sub Y/Y Growth	Rank	Country	2013E Smartphone Subs (MM)	Smartphone as % of Total Subs	Smartphone Sub Y/Y Growth
1	China	354	29%	31%	16	Spain	20	33%	14%
2	USA	219	58	28	17	Philippines	19	18	34
3	Japan*	94	76	15	18	Canada	19	63	21
4	Brazil	70	23	28	19	Thailand	18	21	30
5	India	67	6	52	20	Turkey	17	24	30
6	UK	43	53	22	21	Argentina	15	25	37
7	Korea	38	67	18	22	Malaysia	15	35	19
8	Indonesia	36	11	34	23	South Africa	14	20	26
9	France	33	46	27	24	Netherlands	12	58	27
10	Germany	32	29	29	25	Taiwan	12	37	60
11	Russia	30	12	38	26	Poland	11	20	25
12	Mexico	21	19	43	27	Iran	10	10	40
13	Saudi Arabia	21	38	36	28	Egypt	10	10	34
14	Italy	21	23	25	29	Sweden	9	60	16
15	Australia	20	60	27	30	Hong Kong	8	59	31

2013E Global Smartphone Stats: Subscribers = 1,492MM Penetration = 21% Growth = 31%

KPCB

Note: *Japan data per Morgan Stanley Research estimate. Source: Informa. 40

Figure 2: *Detailed breakdown of smartphone users by geography. (Source: Internet Trends by Mary Meeker at KPCB).*

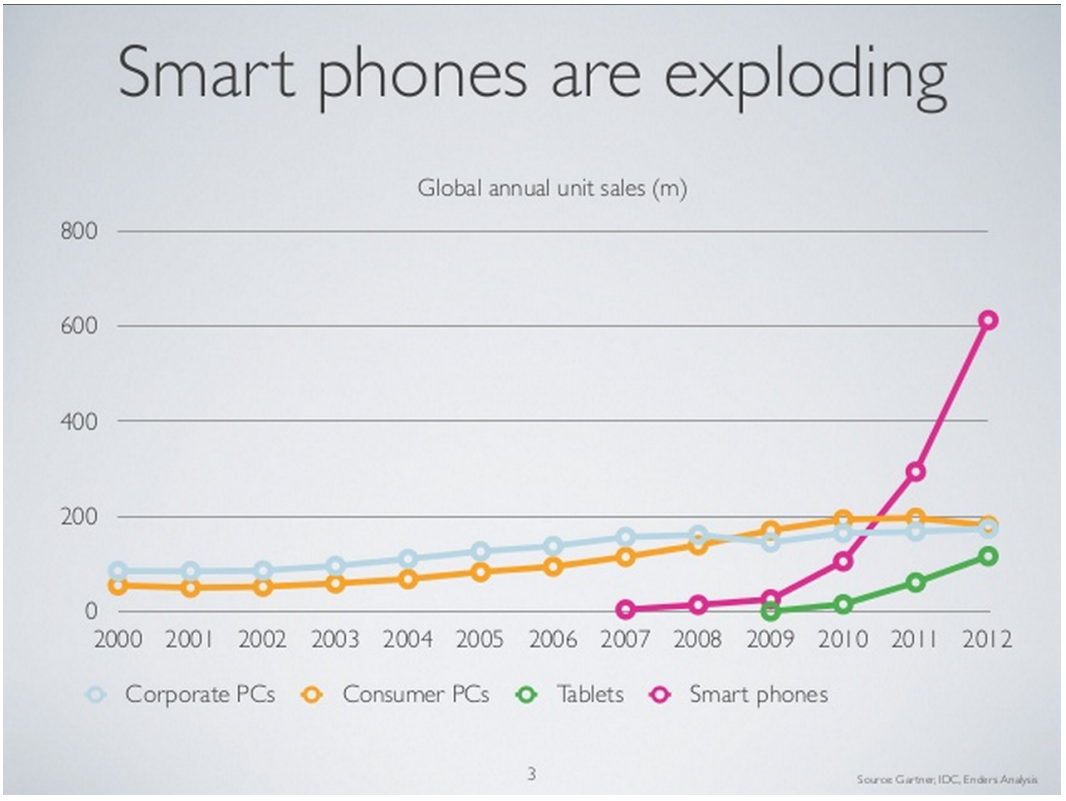


Figure 3: *The installed base of smartphones and tablets has roared past that of PCs. (Source: Benedict Evans, BEA May 2013).*

Is Mobile Simply a Fad?

There are certainly technologies that were in retrospect dead ends or fads; XHTML 2 and (more arguably[1]) Second Life fall into this category. And you will hear programmers of a certain stripe complain about how HTML5 and other browser technologies are just reinventing the same tools that were present on the desktop 15 years ago. "Wow", they will say - "now I can rotate a sphere in a browser window rather than a desktop window!" While true, this minimizes the novel aspects of web apps, prime among them being user convenience. Web apps require zero installation, are always up to date, have a familiar browser interface, can be integrated with other apps, and (if well-designed) have bookmarkable and shareable URLs. Putting Quake in a browser with zero software to install results in a qualitatively different app, if only for the rapidity of viral spread; the time delay between starting a game and pasting a link to said game in an email or Reddit thread is much shorter than the time to get ten friends to buy and install a desktop application.

Similar complaints are often voiced regarding mobile apps, as many mobile apps are indeed simply stripped down desktop apps, with fewer features and often no obvious reason to be mobile - aside from convenience. But when it comes to the marketplace, that convenience is everything. Here's Yishan Wong, Reddit CEO and former Facebook Director of Engineering on the topic:

> **Why is mobile the future?**
>
> *Q: So all I hear is that the mobile internet, mobile devices, geolocation etc, is the future. The web, and the web on desktops is dead. When I ask people why, they say 'coupons'. You check in at a bar and then send you coupons. A coupon driven revolution or paradigm shift seems like weak-sauce to me. Can someone tell me why mobile is the future without mentioning coupons. Or will the next revolution really be based around deals and monetary incentives?*
>
> Yishan Wong: The answer to this so simple that it boggles the mind.
>
> The reason is that human beings are mobile.
>
> We are not sessile organisms confined to a single location for all our lives. If you look at every other technology ever developed, you'll see that the ability to carry it around and use it in arbitrary locations vastly increases its utility and versatility.
>
> Therefore, the emergence of (1) sufficiently powerful computing processors to run a client device and (2) always-on low-latency global coverage to link these clients to an arbitrary cloud computing and storage resource means that computing power is now mobile. Computers are an incredibly useful tool, but now that you can carry a powerful one around with you, thus removing the constraint that you sit in a single location if you want to do any computing and furthermore, extending the available things in the world upon which you can apply computation.

[1] This is arguable because it is possible that Oculus Rift may be to Second Life what the iPhone was to Pandora: a new technology that gives a second shot in the arm to an idea that was too early.

> So: the reason isn't "coupons" at all - not even close. Coupons is one extremely minor and narrowly specific application of computing mobility. The real reason mobile is "the future" is much bigger and abstract: every tool of value to humans is and always was mobile to begin with, and if it had to start out in its early phases as something large and cumbersome, that's merely a temporary state upon which there exists a natural force that would immediately make it mobile if only it could be made lighter and more portable.

The Mobile Present

We should make this clear at the outset: for the purposes of the *class*, we'll be building a basic crowdfunding webapp which uses Bootstrap's responsive features to get a reasonable level of mobile-friendliness. However, for the purposes of your longer term business you may indeed want to build a native app. So it's worth surveying the state of mobile computing today.

Google/Samsung/Apple and Android/iOS

We begin with the obvious: the dominant players are Samsung and Apple (on the hardware end) and Google's Android and Apple's iOS (on the software end). After them, Amazon's Kindle Fire tablet is worthy of mention, as is the hotly anticipated Firefox Phone. Of these players, iOS is still on top from a monetization standpoint, as reflected in the fact that as of about a year ago, YC startups still developed for iOS first. Here's Paul Graham on the topic:

> To most startups we fund, iOS is way more important. Nearly all build for iOS first and then maybe one day port to Android. There are a few exceptions like Kyte who use Android to do things you can't do on iOS. And of course Apportable has been very successful auto-porting iOS apps to Android.

However, while Apple has cumulatively paid developers more than $10 billion over the five year lifetime of iOS, Google is gaining major ground. Android has long been the frontrunner in the mobile space by handset volume, is about to catch up on downloads, and has closed the app revenue revenue share gap from 81%/19% to 73%/27% in the span of a year. Android is also starting to roll out new web services (e.g. gesture typing, Google Now) that iOS can't seem to match. While cliche to state it, with Jobs' passing it is likely that the long-term game is going to to Google.

It's also important to know how these players make their money. Apple makes about $1B in profit annually from the App Store while Google did $17.5M in May on $350M in revenue ($210M profit on $4.2B revenue annualized). However, Google's real mobile monetization is through mobile ads (roughly $8B annual revenue as of 2013) and Apple's real monetization is through first party hardware sales (roughly $160B annualized revenue and $40B profit as of Q2 2013, of which the majority comes from iPhone and iPad sales). Samsung is actually the party which is making money from Android hardware sales, toting up an impressive $8.3B in profit on $50B in revenue in Q2 2013 alone.

Thus right now the revenue from the mobile hardware buildout currently leads that from the software buildout by a significant margin. It's almost tautological that this is the case -

one needs devices in people's hands before one can install software - but this lag still means significant opportunity. For example, Kleiner announced a fund of $200M for iPhone software when the App Store launched in 2008; that looked aggressive at the time, but they may make that back on Square alone, alongside their other mobile bets on Shopify, Path, and Flipboard (all much-needed wins after a decade of cleantech losses).

Mobile Cannibalizes Hardware while Growing Software

Perhaps the best way to conceptualize why the mobile hardware buildout is epochal for the software industry is that hundreds of millions of new customers bought handheld computers while they thought were buying phones, thereby vastly increasing the theoretical market size for internet-connected computers and applications. Moreover, in addition to simply being a mobile computer, smartphones in particular can be thought of as replacing (Figure 4) a wide swath of specialized devices (1, 2, 3, 4, 5, 6), and enabling new modes of user interaction: the phone as a remote, the phone as a docked device, the phone as a port for more specialized sensors or devices.

The combination of these trends means trouble for traditional purveyors of standalone consumer electronics hardware (e.g. Garmin, Sony) and tremendously more reach and distribution for software developers. An increasing number of hardware device equivalents (Figure 4) are now available at all times, and interesting apps like Square and Clinkle combine these built-in hardware APIs in clever ways. Square used the fact that the iPhone headphone jack was actually an active port that received a signal input from the headphone jack. They used this to make a hardware device that plugged into the headphone jack and sent credit card information over this low-bandwidth channel, thereby creating a cross-device front-facing credit card scanner, avoiding Apple's tax on devices that use the Dock Connector, and spawning a company with a $3.25B valuation. Clinkle hasn't yet been released, but appears to use a smartphone's built-in microphone and speaker as an analog communications channel, thereby sending payment information via ultrasound encoding without the necessity of an internet intermediary. But one doesn't need this level of cleverness to do something of general utility: as a software engineer, the smartphone buildout means you can assume the hardware will simply be an API call away.

Figure 4: *A mobile phone replaces a slew of specialized devices. (Source: devost.net).*

Mobile Constrains Business Strategy

If you are launching a software-based startup today, it is important to think about whether there is *any* way you can put a fundamentally mobile app (native or web) at the core of your service. Those businesses that did not think about this angle up front have been forced to adapt. Apple was one of the very first to see this: they changed their name from Apple Computer right before the launch of the iPhone, to redefine themselves as a mobile devices company. At a smaller scale Taskrabbit (a very useful service for finding temporary labor) has been forced to pivot to compete with Exec, which targeted the same market but focused on a mobile/realtime experience (i.e. get assistance via phone from anywhere with rapid response time). Pandora similarly struggled for years to achieve revenue until the iPhone arrived and supplied the missing distribution they'd needed all along. As Tim Westergren, Pandora CEO noted at the time: "The iPhone changed everything for us. It almost doubled our growth rate overnight. More importantly, it changed way people think of Pandora, from a computer service to a mobile service."

To incorporate mobile into your strategic thinking may seem like a trendy, artificial constraint. It's obvious that mobile apps are useful for transportation, tourism, maps, and anything related to a change in location. Yet what does mobile have to do after all with (say) business expenses or furniture shopping? However, then you start thinking "oh, I'll turn the phone into a mobile scanner for business receipts" and you get Expensify. Or you might reason that "it would be useful to barcode-scan an arbitrary object with your phone to comparison shop" and you get Amazon Flow, which retailers complain is turning their shops into glorified showrooms.

The reason this "think mobile" constraint is actually useful is that if you can figure out a fundamentally mobile angle for your app, you will have come up with something new that was technically infeasible five years ago (before the launch of the iOS App Store) and practically infeasible until relatively recently (now that ubiquitous smartphone penetration in many countries can be assumed). As such your app will[2] invalidate the assumptions of incumbents in the space, granting you a competitive advantage.

The Mobile Future

ChromeOS and Mobile HTML5

There is good reason to believe that Mobile HTML5 is going to wax in importance for app development. The reason is that Google's Sundar Pichai now runs Google Apps, Chrome, *and* Android, after taking over from Android founder Andy Rubin. Google has always thought of native apps as a sort of near-term distraction forced on it by the unexpected success[3] of Apple's iOS. In this vision, HTML5/ChromeOS was a bet on the future and Android was a bet on the present. But the replacement of Rubin with Pichai seems to be an indication that HTML5 is now Google's present, rather than its future. With Apple stumbling and Android gaining, the serious threat from iOS appears to be losing momentum and with it

[2] As Sun Tzu noted, "Thus the highest form of generalship is to balk the enemy's plans; the next best is to prevent the junction of the enemy's forces; the next in order is to attack the enemy's army in the field; and the worst policy of all is to besiege walled cities.". In other words, to win against an opponent with billions of dollars, *always attack assumptions*.

[3] Indeed, even Apple was surprised by this. Few recall this today, but at the launch of the iPhone in 2007 Jobs recommended that people write web applications for mobile Safari. It took a year for them to respond to overwhelming developer demand and build the App Store and the iOS SDK.

the "distraction" of native. Perhaps the most likely scenario over the next 12-24 months is that Pichai will merge Android into ChromeOS (0, 1, 2, 3, 4, 5), likely by doing backwards-compatible rewrites of many Android Java API calls in Javascript. The best analogy for this is MacOS' Carbon to Cocoa transition in the early 2000s. And this means that if you want to play the *really* long game, you might want to start looking at Google Drive Apps, the Chrome Web Store, and the Chromebook Pixel.

The Internet of Things

The so-called internet-of-things (Figure 5) takes mobile even further, moving from items like smartphones or tablets equipped with touchscreens to items which may run mostly unattended and/or be designed to be controlled from a nearby phone's mobile programmable screen. Note that to produce such items again presumes ubiquitous (a) wireless broadband ("internetification") and/or (b) handheld computers as a basic assumption. Another basic premise behind the internet-of-things is that every device will ultimately have an IPv6 address or the functional[4] equivalent. The first internet-of-things devices are already on the market, including Nest (thermometers), Lockitron (locks), and PayByPhone (parking meters); see this post and this image.

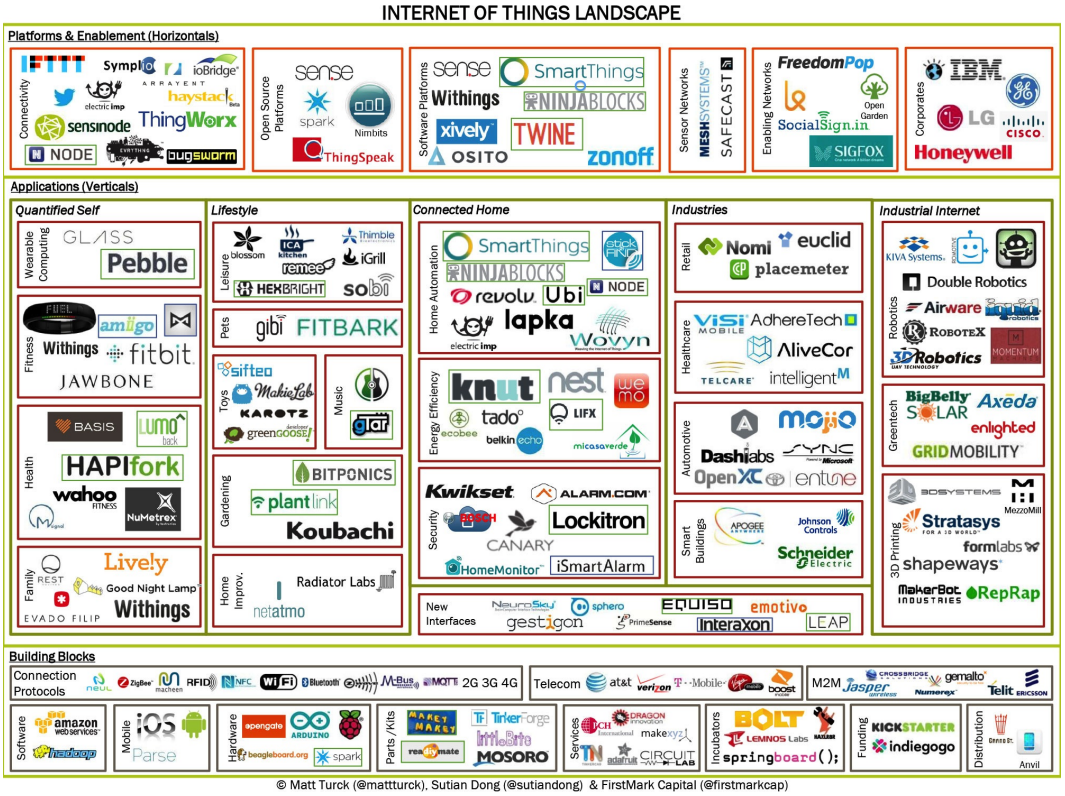


Figure 5: *An overview of companies involved in the internet of things. (Source: Matt Turck, TechCrunch).*

[4]If you are interested in this area, read up on IPv6, Carrier-grade NAT (CGN), and the tradeoffs between the two. In normal circumstances one might predict that providers don't have major short-term incentives to pursue IPv6 rather than CGN. However, due to Google Fiber pushing IPv6, it's a reasonable bet that IPv6 will happen over the resistance of some ISPs.

Quantified Self

Closely related to the internet-of-things is the concept of *quantified self* or QSelf (1, 2, 3, 4, 5). The motivation is that today one can more easily determine what is happening in Bangalore or Budapest than what is happening in one's own body. We can change this by developing sophisticated sensors for the human body that feed data locally into mobile phones or directly to a remote database over the internet. Products in this category include the Fitbit and Jawbone Up armbands, the Withings scale, and the Scanadu Scout vital signs sensor.

While currently quite modest in market size, QSelf technology is likely to ultimately disrupt the way medicine is practiced. As motivation for this statement, consider the contrast between the healthcare industry and the fitness industry. Doctors routinely state that "if fitness were a drug, it would be prescribed for everyone". Study after study rolls in on the benefits of personal fitness; the effect sizes are off the charts (1, 2, 3, 4, 5). Yet your fitness is considered to be your own responsibility: you can join gyms or get good personal trainers, but ultimately the buck stops with you - you need to take the initiative. But consider how that initiative plays out in healthcare. If you come to a doctor's appointment wanting to talk about something you've researched, doctors generally get vexed. Either you are right (and thereby undermining their authority) or you are wrong (and thereby dismissed). This state of affairs is odd: you are with your body for a lifetime, whereas the doctor is only with you for twenty minutes each year. Thus the one area of medicine that really works - fitness - operates quite differently from the rest of medicine in practice.

The fundamental difference between the two areas is the presence/absence of mandated intermediaries. You do not need to pay a physician, an insurance company, a hospital, or (indirectly) a federal regulator in order to step on a scale. But you do need to make some or all of these payments in order to get a prescription medical test. QSelf is going to change all of this, by making many body sensors fundamentally mobile and thereby making it impractical to centrally limit access to diagnostics. We're already seeing this to some extent; for example, a sequel to the Fitbit sleep monitor could enable at-home sleep apnea tests, and the Scanadu Scout or an equivalent will likely replace an expensive trip to the doctor's office to be examined with a stethoscope and a blood pressure cuff. Ultimately this particular mobile technology path culminates in vastly improved self-diagnostic capabilities. And from there? Perhaps even a movement to legalize self-treatment.

Mobile Technology

Let's now introduce two concepts which will be useful for understanding how we can handle the profusion of mobile devices: HTTP User Agents and CSS Media Queries.

Preliminaries: HTTP and User Agents

You've surely heard of the Hypertext Transmission Protocol (HTTP), which is how browsers talk to web servers (among other things). This protocol is a complex beast that deserves its own class, and we won't get into the guts of it here. The main thing you need to know for now is that when you connect via browser to a remote web server, you are sending an HTTP Request and getting back an HTTP Response. An HTTP Request can be thought of as a function call (GET, POST, etc.) that accepts a positional argument (a resource like a URL) and a long list of optional keyword arguments (the HTTP Headers). You can generate HTTP requests with:

- A command line tool (ex. curl)
- A web browser (ex: Chrome)
- A library call (ex: node's restler)
- A web UI (ex: hurl.it)

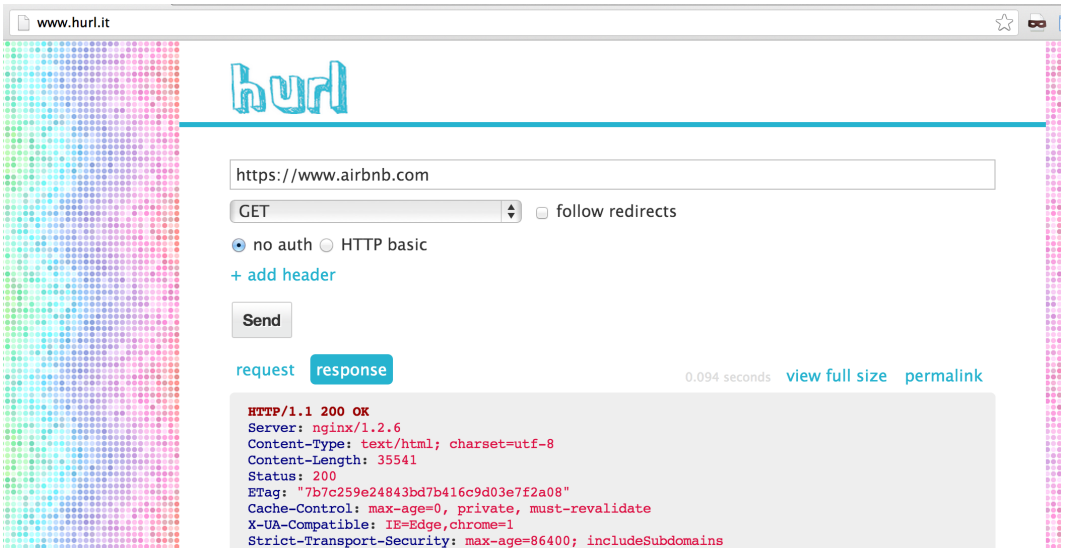


Figure 6: *Here's an example of an HTTP GET request to the resource https://www.airbnb.com.*

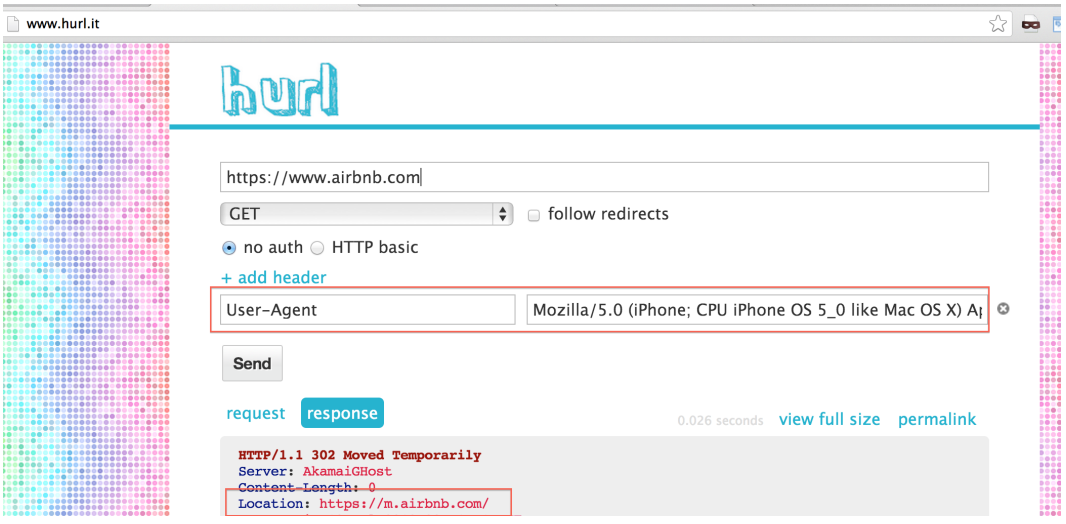


Figure 7: *And here's that same HTTP GET request to the resource https://www.airbnb.com, except with the iPhone 5 User Agent. Note that the HTTP response is different.*

When a web server receives an HTTP request it issues an HTTP response. The exact response can depend upon the time of the day, the user's location, or many other parameters - including the value of the HTTP request's `User-Agent` header (Figures 6-7). The `User-Agent` (UA) is the string that the HTTP client presents as self-identification (examples). It is trivial to spoof the `User-Agent` and thereby pretend that you are a device different from what you are (e.g. Chrome's `User-Agent` Switcher). Nevertheless, on the server side one can do so-called "User Agent sniffing" as a first cut approach to determine what device is connecting to our site. While no-doubt fraught with issues related to the sheer profusion and unreliability of UA strings, this technique is nevertheless used by many major websites (e.g. AirBnB) to serve up different content for different user devices. We should note that many references counsel against UA sniffing and recommend feature detection instead, using a library like modernizr (1, 2, 3, 4). Sometimes feature detection is indeed what you want, e.g. if determining whether the given client supports a particular HTML5 API call, but often that can turn into a bit of a Rube Goldbergish way to simply specify that you have a custom site for (say) iPhones or iPads. Use your judgment here.

Preliminaries: CSS Media Queries and Responsive Web Design

We talked about Cascading Style Sheets (CSS) a bit in Lecture 6. In Homework 4, CSS is used not just to control typography, for layout, and for graphics, but also to detect the screen width and use this to apply styles conditionally. This is called a *media query*; it's basically a rudimentary if/then statement which executes different CSS code based on the width of the current browser window. Media queries are the basis for Responsive Web Design (RWD), where the page layout adapts to the screen width of the viewing client. See examples here.

The Mobile Problem: Diversity of Devices

With these preliminaries, we can discuss an important issue in "going mobile": namely the multiplicity of screen widths (Figure 8) and OS versions on Android (Figure 9), and more generally the wide array of potential API clients (Table 1).

Table 1: *The general mobile API client fragmentation problem. From the point of view of API-based design or service-oriented architecture (SOA), mobile means not just mobile devices, but the set of potential clients/distribution targets for your application/content. Note that for several of these targets (iPhone, iPad, Android) one may also need to design an alternate layout of the page for portrait and landscape mode.*

API Client	Client language/libraries
Desktop web	HTML/CSS/JS
Mobile web	HTML/CSS/JS
iPhone	Objective-C/iOS
iPad	Objective-C/iOS
Android devices	Java/Android
Native Mac app	Objective-C/Cocoa
Native Windows app	HTML/CSS/JS/Metro UI or XAML
Command line	Node, Python, Ruby, Java, ...
3rd party API client	Node, Python, Ruby, Java, ...
Twitter cards	Twitter API

Continued on next page

Table 1: *The general mobile API client fragmentation problem. From the point of view of API-based design or service-oriented architecture (SOA), mobile means not just mobile devices, but the set of potential clients/distribution targets for your application/content. Note that for several of these targets (iPhone, iPad, Android) one may also need to design an alternate layout of the page for portrait and landscape mode.*

API Client	Client language/libraries
Facebook Open Graph	FB API
Google Rich Snippets	Google Microformats
Google Glass	Glass Cloud API
iWatch (?)	Likely Objective-C/iOS
iTV (?)	Likely Objective-C/iOS

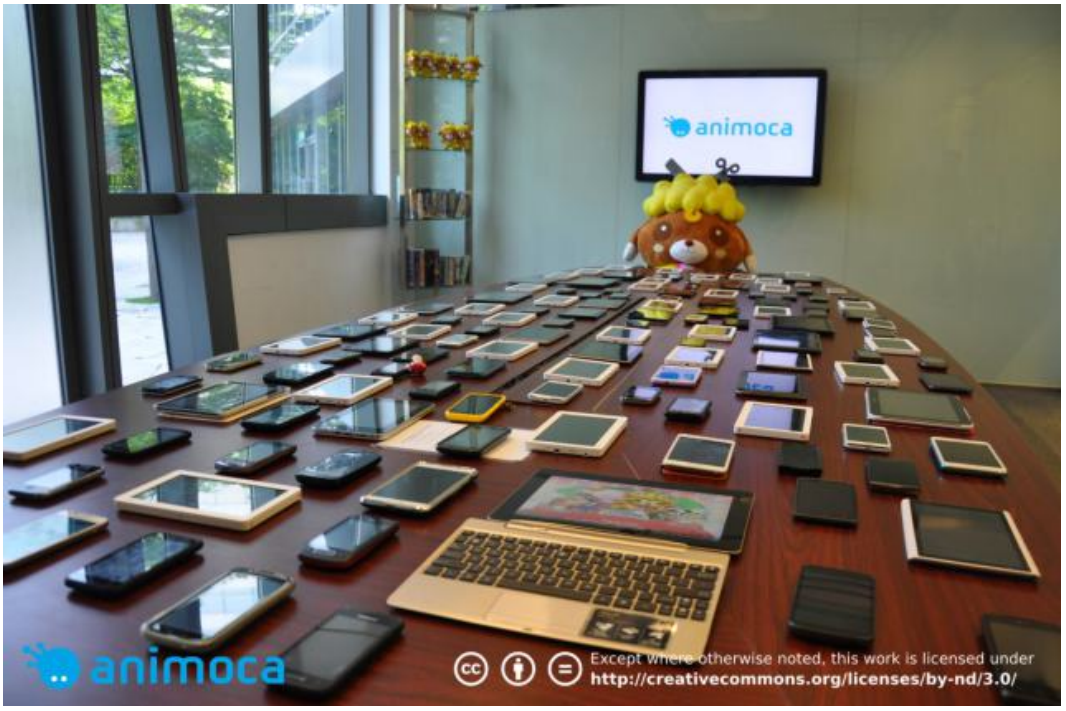


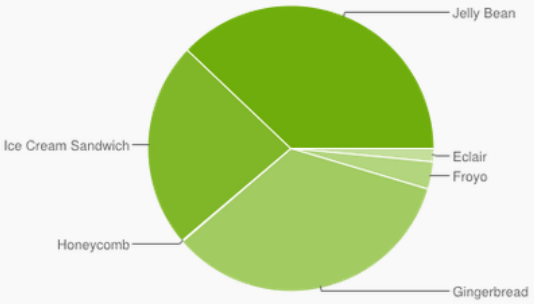
Figure 8: *The Android OS screen fragmentation problem. (Source: Techcrunch).*

Platform Versions

This section provides data about the relative number of devices running a given version of the Android platform.

For information about how to target your application to devices based on platform version, read Supporting Different Platform Versions.

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.4%
2.2	Froyo	8	3.1%
2.3.3 - 2.3.7	Gingerbread	10	34.1%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	23.3%
4.1.x	Jelly Bean	16	32.3%
4.2.x		17	5.6%



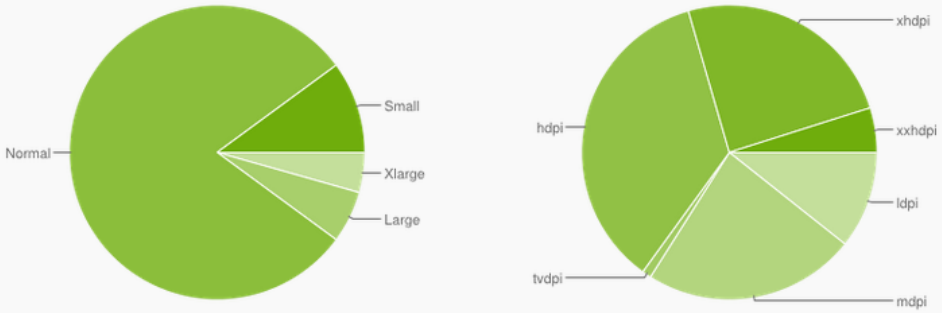
Data collected during a 14-day period ending on July 8, 2013.
Any versions with less than 0.1% distribution are not shown.

Screen Sizes and Densities

This section provides data about the relative number of devices that have a particular screen configuration, defined by a combination of screen size and density. To simplify the way that you design your user interfaces for different screen configurations, Android divides the range of actual screen sizes and densities into several buckets as expressed by the table below.

For information about how you can support multiple screen configurations in your application, read Supporting Multiple Screens.

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	9.9%			0.1%			10.0%
Normal	0.1%	16.0%		34.9%	24.0%	4.9%	79.9%
Large	0.6%	3.2%	1.0%	0.4%	0.5%		5.7%
Xlarge		4.1%		0.2%	0.1%		4.4%
Total	10.6%	23.3%	1.0%	35.6%	24.6%	4.9%	



Data collected during a 14-day period ending on July 8, 2013
Any screen configurations with less than 0.1% distribution are not shown.

Figure 9: *The Android OS fragmentation problem (Source: Google.)*

While a bit of a pain, the situation is not as complicated as these figures and tables may make it out to be. From a business perspective, you need to consider whether you will be optimizing for sheer global availability (in which case a mobile web app with RWD is the first choice), for maximum current App Store revenue (in which case iOS-first is still the best), or for the maximum number of handset users and potential future app revenue (in which case Android-first may be worth a bet).

From a technical perspective, there are essentially two options for dealing with client diversity in the context of mobile web apps. At one extreme one can use the one-size-fits-all approach of responsive web design. In this case, the server returns the same HTTP response for all clients, with CSS media queries in the body of the response used to implement some conditional logic (e.g. hiding or showing certain divs on a phone or tablet). At the other extreme the server can return *different* HTTP responses for different User Agents, as specified in the headers of the HTTP request. This can be used to deliver custom mobile-optimized sites and is what Facebook, Google, AirBnb, Twitter, The Guardian, TechCrunch, and now Github are doing (often via a URL like `m.example.com`). An even more labor-intensive version of this is to build both a mobile web app and a native app for each client.

The second approach of custom sites for each client is significantly more expensive, but can be partially facilitated with a so-called API-based design or Service-Oriented Architecture (SOA). In this setup the core of the application is an API calling a database. All clients (internal command line tools, the mobile web, and iOS/Android) then simply call this API to create/read/update/delete objects and then render them using the tools of their native environment: e.g. ASCII for the command line, HTML for the mobile web, iOS widgets for iOS apps, and so on. (Table 1) That said, even with this kind of approach one needs engineers to constantly maintain the templates for each and every device (and in both landscape and portrait mode), ensuring that they don't break or subtly lose functionality as operating systems are updated and new devices come out.

While Facebook can afford this kind of investment, for a small company it is infeasible to support many different clients. And it is usually unnecessary: picking one platform and using it to prove out your market is probably the best idea. Steve Jobs' heuristic was to pick a "technology in its spring", a technology which is fresh and growing yet has been used in production applications by at least a few people. One way to measure this is by comparative Github forks/stars, Stackoverflow activity, Google trends/search data and the like. Obviously, the ideal first platform will change over time; for example, Instagram timed their pure iOS bet perfectly, but a company that was doing something similar today might seriously consider doing Android first (or perhaps even getting a Google Glass dev kit).

Mobile Frameworks: Pure HTML5 vs. Cross-Compilation

Let's make one point of clarification up front, regarding the distinction between "libraries" and "frameworks". While a little fuzzy, the difference is that a library is built to solve a single problem. You just `import` or `require` it and then use it in conjunction with several other libraries. By contrast, a framework is a collection of libraries with its own internal data structures and conventions that ends up governing the way you approach a space. When it comes to web development, the advantage of using a set of libraries is that you have great flexibility; the disadvantage is that in filling the gaps between these libraries you will end up writing your own framework. Conversely, the advantage of a framework is that it's "batteries included", but the disadvantage is that it can be hard to do something that greatly violates

the underlying assumptions of the framework.

For the purposes of the class we'll be using the Bootstrap 2 framework, because it's widely used, gets you on base quickly on all platforms, and is very flexible and skinnable as frameworks go (e.g. wrapbootstrap.com and bootswatch.com). Moreover, by using Bootstrap 2 it'll be relatively easy to migrate your app to the new Bootstrap 3, which is now a true mobile-first framework. Two possible alternatives to Bootstrap 3 are JQuery Mobile (JQM) and Sencha Touch (ST). Both of these are pure mobile-optimized HTML/CSS/JS frameworks which don't require knowledge of iOS and Android development. The sites built with these are still viewable in a desktop browser but are optimized for the mobile web (example). With some effort you can usually mix and match components from JQM or ST into a primarily Bootstrap app (or vice versa), but it's generally easier to just stick with one[5] of these frameworks.

One might also consider using some of the cross-platform frameworks like Phone Gap or Appcelerator Titanium. These frameworks let you write an app in HTML/CSS/JS and then cross-compile it to native Android and iOS apps (and also to Windows Phone, Blackberry, etc.). While a valiant effort in many respects, these kinds of frameworks are in our view a bit of a no man's land between a pure web app vs. a pure native app. An RWD app with Bootstrap 2 will get you "on base" on each platform with something serviceable, while a pure native app will get you high performance, access to the hardware, easier debugging, and native-looking widgets. But a cross-compiler framework is yet another thing to learn, and won't usually give you the same snappiness of a native app. It's arguable as to whether it's really better than (1) just biting the bullet and going native or (2) alternatively, taking your initial RWD web app and making a mobile-optimized 2.0 version.

Now, the choice between these two is the question of whether or not you can get a mobile HTML5 app up to the same level of performance/snappiness as a native app, which is perhaps the most important strategic debate in mobile development today. Facebook famously tried and failed to do a pure HTML5 mobile app, at least by their standards and with their approach. However, this impressive demo/post by Sencha Touch does indicate that significant performance improvements over FB's initial version are feasible. Still, LinkedIn recently announced that they also switched back from mobile HTML5 to native due to memory issues and widget snappiness. In general it is fair to say that it is still nontrivial to get a mobile web app up to the same level of snappiness as a native app, but at the same time a mobile web app gives you a reasonable first presence on every device and is quick to build.

A Mobile Solution: RWD first, Native immediately after, and then use logs

To recap: we're targeting the mobile web for this class because (a) it is relatively easy and will get you "on base" for each platform and (b) it will be useful no matter what you do. However, it should be explicitly noted that even if ChromeOS and mobile HTML5 are the future, the present is still very much native. If you really do want to build a mobile app, you'll want to learn either Android or iOS well, with the mobile web app as an initial stopgap. That said,

[5]Note however that Bootstrap 2 would be best considered a CSS framework, and is meant to work with JQuery (a JS library). You can also easily combine Bootstrap 2 with a frontend JS framework (like Backbone or Angular), and a backend web framework (like express for node.js). That is, Bootstrap focuses on the CSS styling of widgets as opposed to their behavior and how they move data around the page (frontend JS) or back and forth from the server (backend framework). By contrast, JQuery Mobile and Sencha Touch have opinions about how data is moved around the page and maintained within widgets (see e.g. the JQuery Mobile AJAX section). Does all this sound complicated? Yes, welcome to the fast-moving world of web development. You may enjoy this rant.

here is a reasonable approach to get started with an app as of mid-2013.

1. Start with a responsive web design app, built on top of an API.
2. Don't try to build an RWD framework yourself; this is an enormous effort in its own right. Use Bootstrap 2 (or if you want to be aggressive, the pre-release Bootstrap 3). You might also consider Zurb Foundation.
3. Theme your Bootstrap app with an off-the-shelf theme from wrapbootstrap or bootswatch or the like. Customize with Google Fonts or get a designer from dribbble.com or 99Designs.com to do more in-depth customization.
4. Now deploy your site and get some users.
5. Analyze your log data very carefully, looking at `User-Agent` in particular to determine what devices your customers are using.

This approach will let you up and running quickly with something which is functional on virtually every platform. Then you can use the log data with your business strategy to prioritize further targets. Alternatively, you can launch the RWD app and then go with (say) an iOS or Android app, using the log data to prioritize your second target (depending on your product, that might even be IE clients rather than iOS clients). The use of an API at the core of the site will then allow you to generalize to new platforms over time (e.g. Google Glass or Apple's Watch, as well as embedding your app in Twitter or FB).

Mobile Constraints

It's finally worth noting that there are a variety of new issues that crop up with mobile.

- *The network is unreliable.* While we're moving towards an age of ubiquitous wireless internet, we aren't fully there yet. Apps should be designed to work with flaky connections or at least diagnose and display that they are offline. (See: "The Fallacies of Distributed Computing")
- *File sizes need to be small.* Once you get things working, you'll want to start aggressively removing unused JS functions and CSS styles from what you're serving to mobile clients as they are often on low bandwidth connections (e.g. 3G or 4G rather than Wifi).
- *Debugging requires logging.* You want to log aggressively and provide user bug reporting. The nature of mobile device diversity is that you will not be able to anticipate up front all the odd issues that will arise in the field. Do make sure to get your users' permission to send these log files back to your server periodically.
- *Minimize user input.* You should use passcodes over passwords, swipes over typing, and autocomplete over form filling. A great example of this is the Google Maps autocomplete field. It replaces 5-6 fields (Address 1, Address 2, City, State, Zip, Country) and includes error checking to boot.
- *Minimize time to result.* Your app should boot quickly (difficult with an RWD app) and offer big buttons with common use cases.

In some ways the constraints of a mobile environment - where every byte matters, and network connections can be as frustrating as 56k modems - warp us back to 1995 when the internet was young. But bear in mind that was a time of great opportunity.