Web v. Native

Lecture 14

Warning: Stale Data!

- Data on application usage is getting harder to obtain because it has significantly more business value than it used to have (and thus, is no longer free).
- Thus, some of these slides use older data that may not be as accurate as it once was.
- Where possible I've tried to use the most recent data available.

Web Apps vs. Native Apps

- **Mobile Web Applications**
- Web applications; reside on a web server
- Designed for mobile device screens
- Run on the devices' browsers
- Require an Internet connection to use
- Very portable; work on any device with a browser

- **Native Applications**
- Like regular applications that run on desktops and laptops
- Designed to take advantage of the phone's hardware
- Run directly on device
- May run without any connectivity after initial download
- Platform specific unless a generic toolkit and framework is used

5 Concerns about Mobile Applications

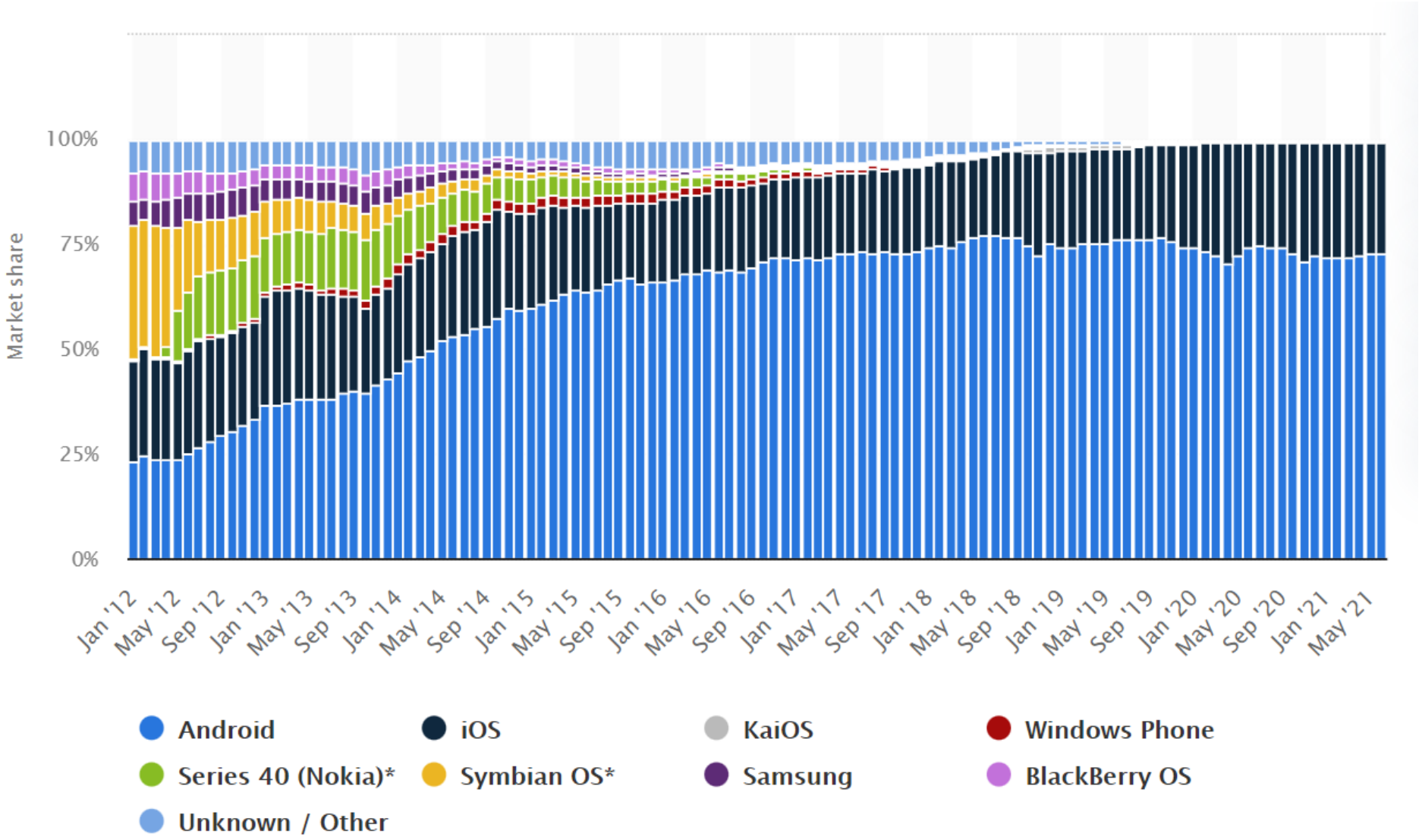
1. Fragmentation
2. Web Leverage
3. Control
4. Consumer Expectations
5. Ubiquity

Fragmentation

The problem:

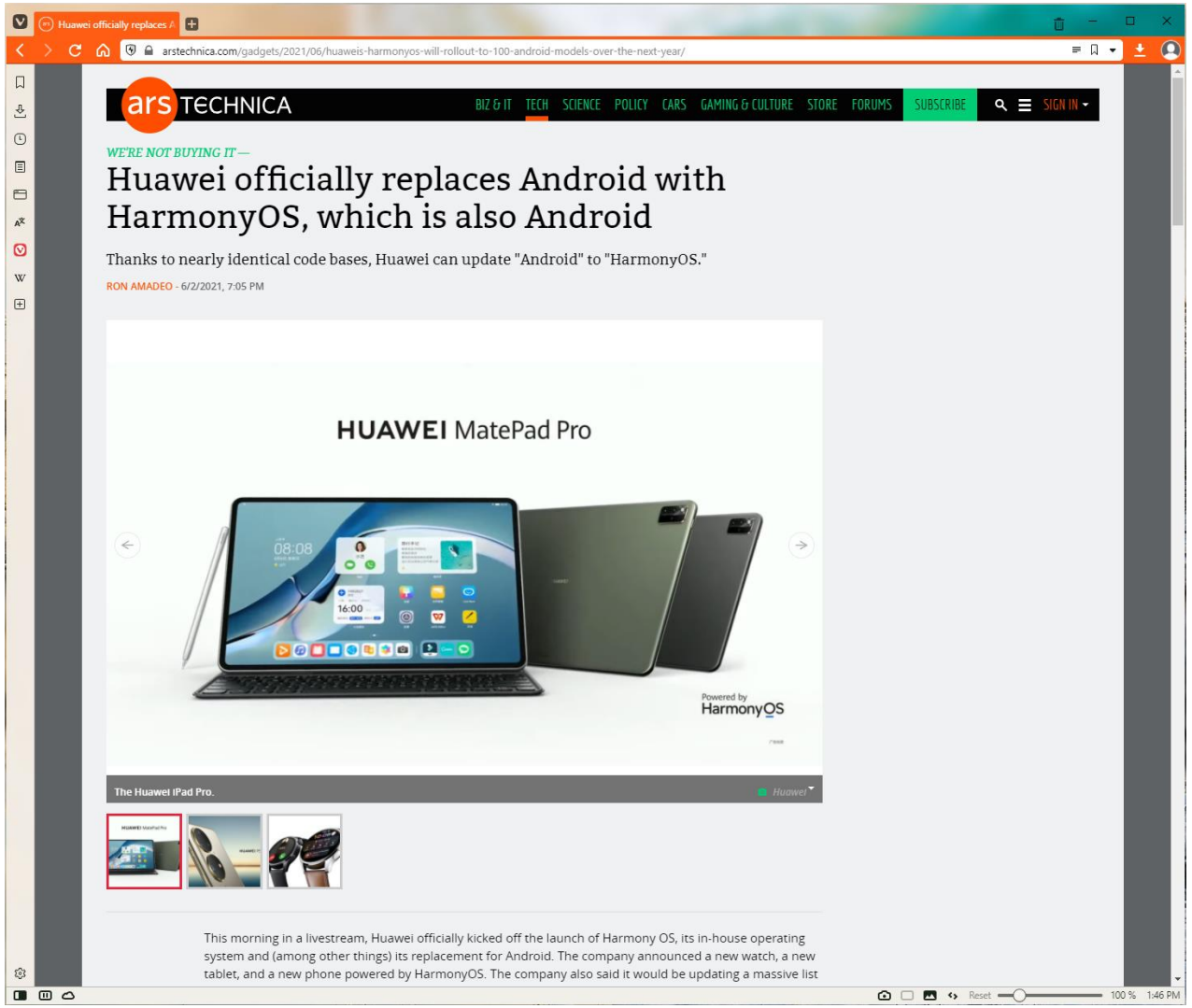
- Historically there were many different platforms for mobile applications.
  - E.g., iPhone, Android, Blackberry, Windows CE.
- Native applications (typically) need to target specific platforms and only run on that platform. If you want to support 4 platforms you need 4 versions of the code.
- Which platform do you choose? (Not too difficult now there are only 2 major platforms.)
- Issue 1: Android uses Kotlin while iOS uses Swift – not very compatible.
- Issue 2: There is more than one version of Android! (e.g., Fire OS on Amazon devices, Harmony OS on Huawei).

Mobile OS Market Share (International) Jan 2012 – June 2021



Not all Androids are created Equal

- See: https://en.wikipedia.org/wiki/List_of_custom_Android_distributions
- Vendors are starting to create their own variants of Android
- Amazon has been doing this for years with it's Fire OS for Amazon phones and Kindle "Tablets."



Fragmentation: A Summary

- Android and iOS devices represent 99.7% of the current market

Only two viable platforms: iOS and Android.

But ... Android is no longer just “an” operating system,
it is a family of similar operating systems.
There could be problems down the road …

Web Leverage

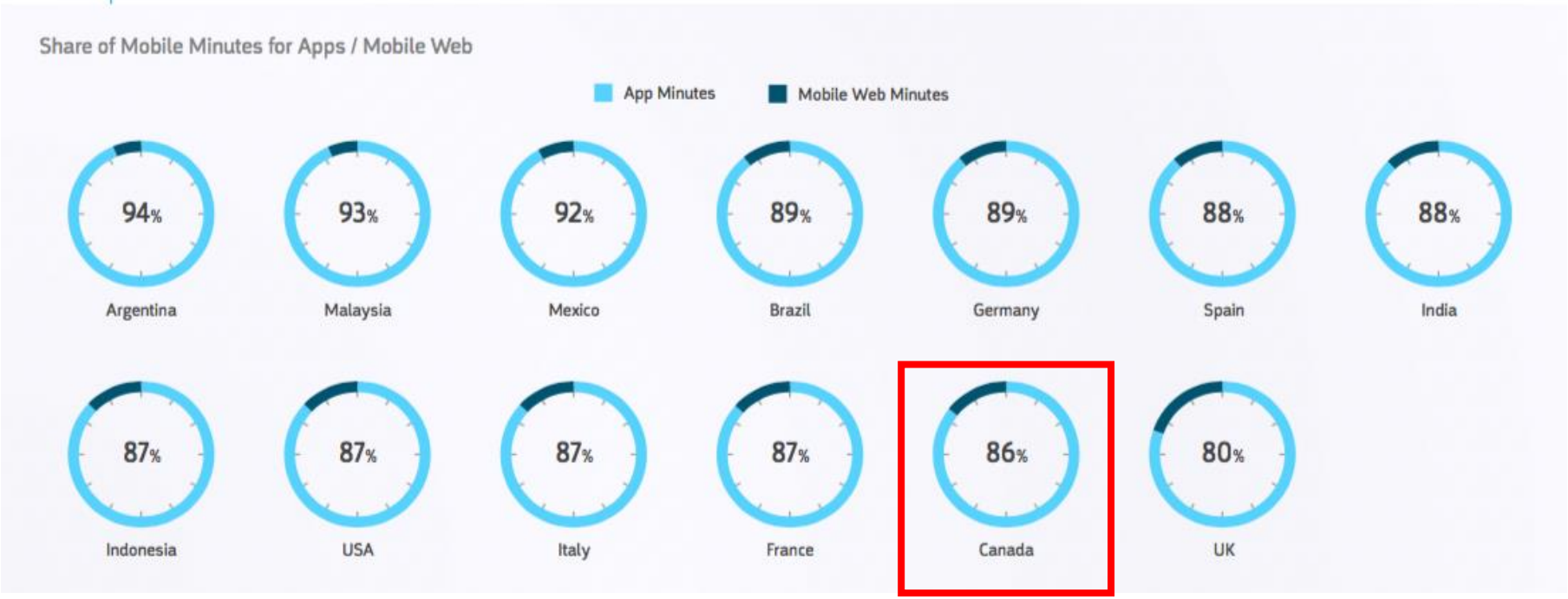
- **Hypothesis:** Most technology is moving over to the web so linking a mobile application to the web allows developers to leverage existing web resources.

- **Contributing Facts:**
  - There is an abundance of content on the web.
  - There is an abundance of technology for the web.
  - All you need to run web applications is a browser, which all mobile devices have pre-installed.
  - Using the web thus avoids issues with fragmentation and creates device independence.

"The Internet in your pocket."

Apps drive dominant share of mobile time in all markets

More than 80% of mobile minutes in all markets are spent on apps



Users prefer using mobile apps on their phones over browser-based apps.
Web leverage is NOT a valid argument to support writing browser-based apps.
- Usable without a connection, reduced data costs, native UI preferences

Improved Control

- **Concern**: Native application distribution will be controlled by platform vendor (iStore, Google Play), not application developer.

- **Examples**:
  - App stores are gatekeepers to all new applications.
  - To release an application, a developer must:
    - Create and upload the application to an app store.
    - Get the application approved for release.
    - Wait for users to download the application (not guaranteed).

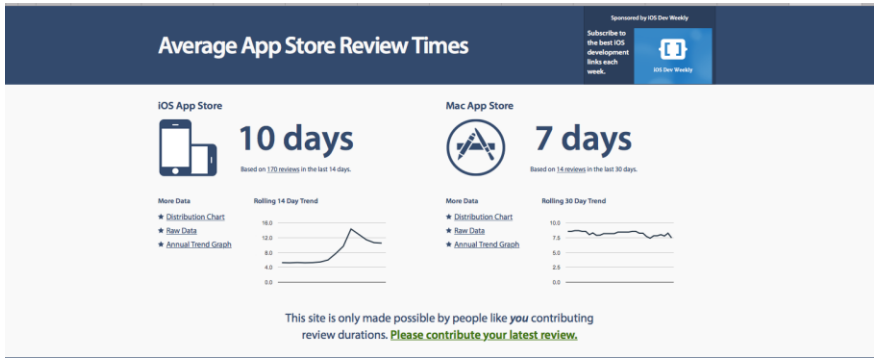
- **Why?**
  - Purpose of mobile applications is to get users to buy more mobile devices. Apple and Google want to control the quality of apps as application quality influences perceptions of device quality.

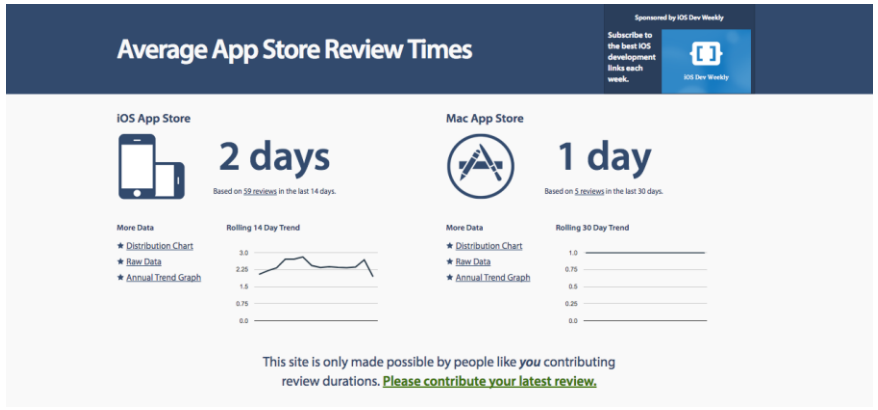
Possible Control Issues

Issues with lack of developer control:

- Deployments, updates, changes take a long time if you have to wait for users.
- Recalls, or retractions may be impossible.
- Your platform may be shelved at any time (e.g., What would happen if Google replaces Android with something new?)
- Apple/Google can change libraries, languages, hardware at any time and break your application (e.g., Google moved from Java to Kotlin, Apple moved from Objective C to Swift).

Average Review Times (iOS Apps)





These days, it takes **about one day** for reviews of an app or update to appear for an app submitted to the iOS or Mac App Store. It doesn't matter if it's a new app or just a minor new version. It doesn't matter whether it's paid, free, uses in-app purchases, or subscriptions. It doesn't matter if the developer is a small company or a multi-national corporation. It doesn't matter what category of the App Store it's listed in. It simply takes **about one day** for the vast majority of apps to be reviewed. (Retrieved Sept. 2019)

If an app is reviewed that means it's been seen and tested by reviewers. Using an app store doesn't slow release and deployment very much now.

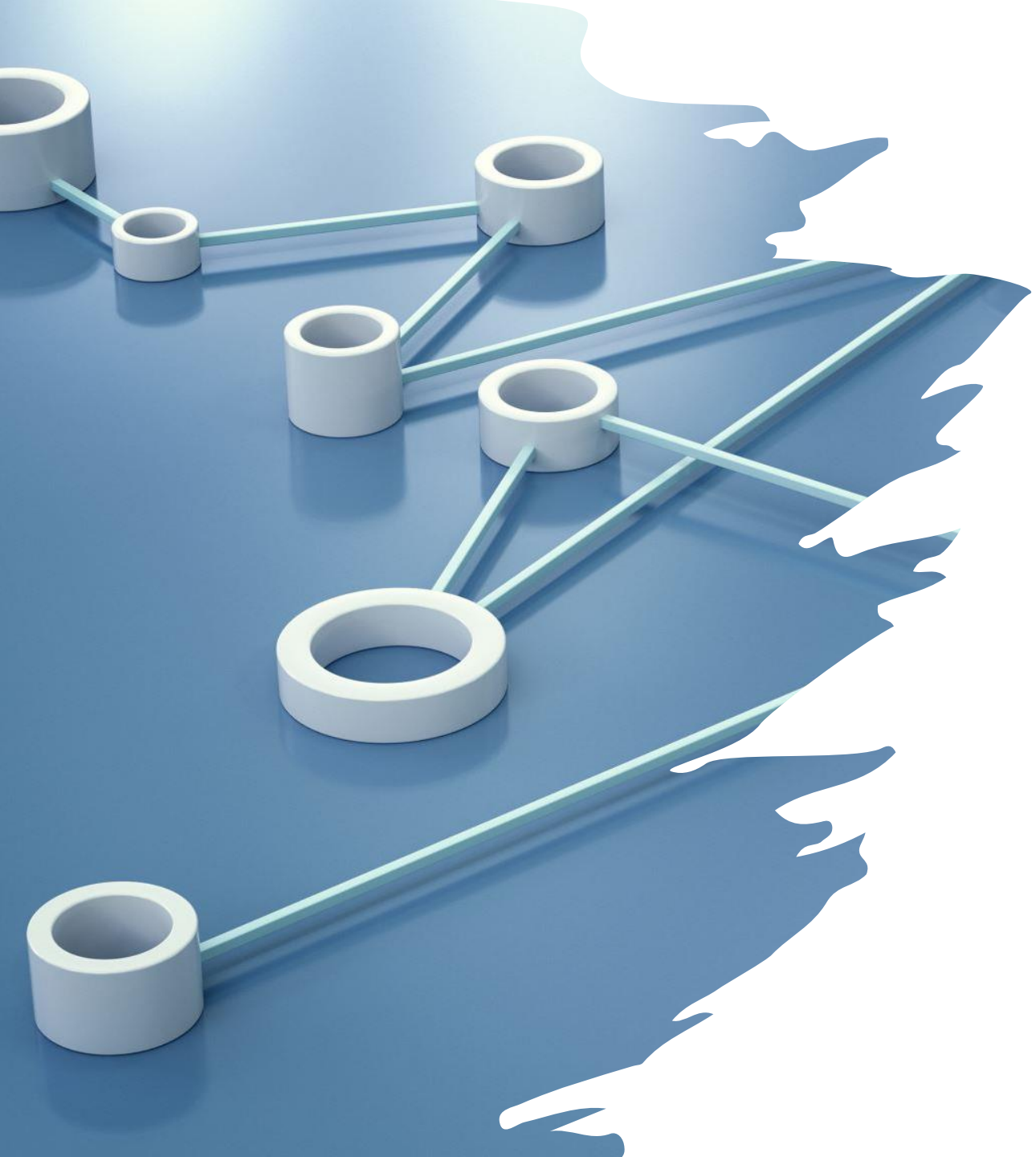
Source: appreviewtimes.com

Web-Based vs. App Store Distribution

- It is Marketing and Advertising that most significantly affects distribution times.
- App stores are easy to find and search.
- App stores tend to be trusted.
- App stores tend to have higher standards and take some control away from the developer, but only minimally and often beneficially.
- App stores do not cause vendor lock-in, as this is a property of the platform, not the distribution system.

Loss of control in using App Stores is minimal at worst

and virtually non-existent in most cases.



Aside:
Career Lock-In

- Do platforms play a key role in your career decisions?
- Why?
- How do you select one platform over another?
- Is this reasonable and practical, or simply a necessity?
- Does Google, Apple (and to some extent Microsoft, Oracle, etc.) have an unnecessary or undesirable control over your future?

Consumer Expectations

Consumers want things to just work.

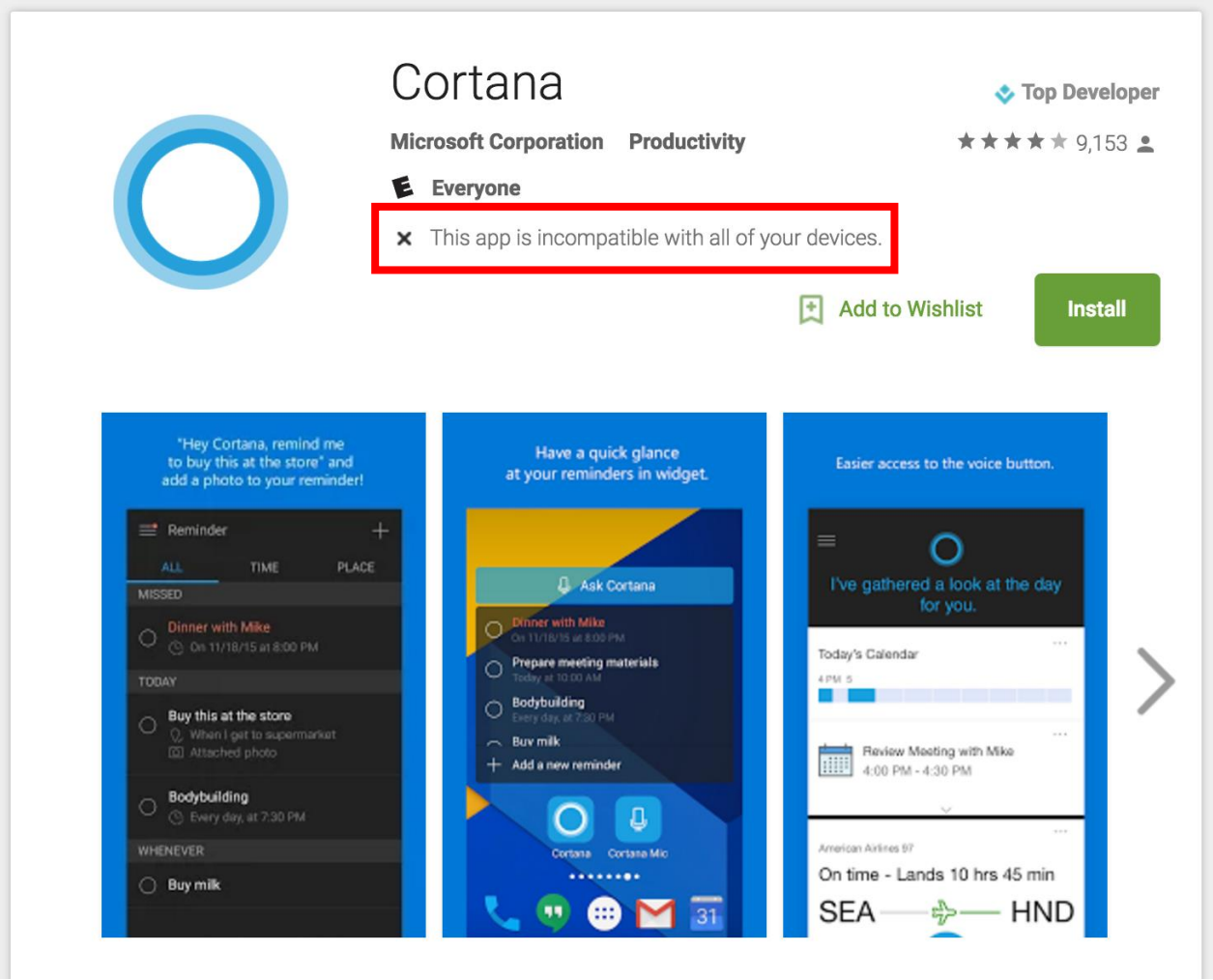
- Applications should just run (and download).
- Users don’t care whether their device is compatible.
- Users don’t want to figure out if the application is safe or what permissions to allow.
- Users don’t want or expect software to change because they change devices.

The customer’s perspective often tends to be:

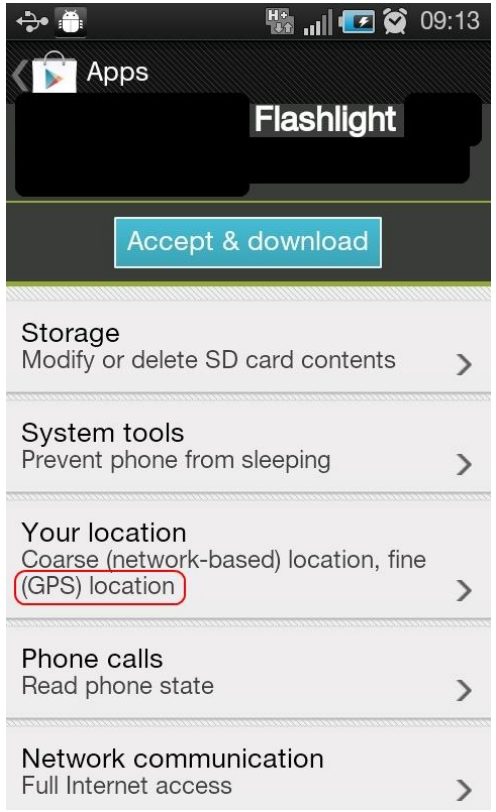
“I spend a lot of money on a mobile device; I want content (software) to effectively support it.”

“Smart phones should be like cars. You don’t have to learn how to drive a Ford, Fiat, or Toyota differently, thus why should I have to learn how to use all the different phone/app types?”

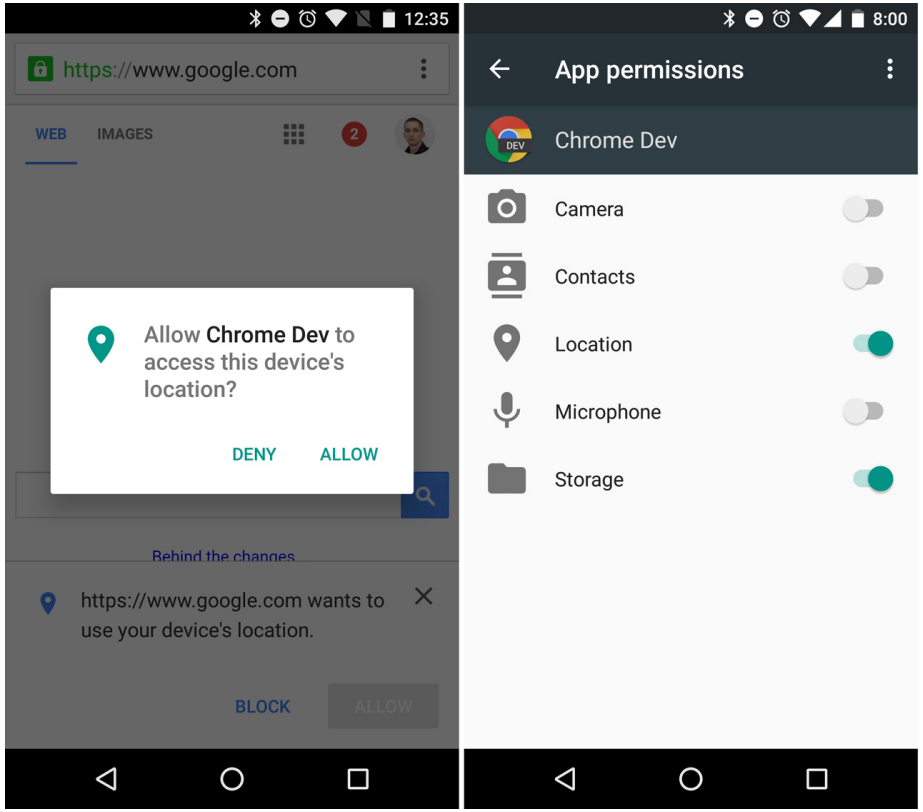
Example: Consumer Expectations



Example: Consumer Expectations



and



Why would a flashlight app need your GPS location?
How does a user benefit from sharing this data?

Consumer Expectations

- In general, it is usability which matters, not platform, interface, or distribution method.
- Users want usable software and a lack of usability is one of the primary ways you can violate their expectations.
- Users want things to make sense.
- Users want things to be easy.
- Users don’t want to waste money (on data, on 2 versions of the app – iOS & Android).

If you design highly usable software, the platform (web or native) isn’t really that important.

Using a browser doesn’t increase usability or help satisfy consumer expectations.

Aside: The Update Dilemma

- Some users would never update an application if it was not imposed on them.
- Developers often feel that it is important for users to have the most updated version (though this may not match all users' desires).
- To combat a "lack of update," developers implement auto-update to automatically update software so that users do not need to do so.
- Some users dislike auto-update! It uses data, can slow down their phone, and creates the feeling that control has been taken away from them.
- Is it better to upset some users with auto update or to have some users with older versions of the software?
- This dilemma doesn't exist with web applications.

Ubiquity – “Run anywhere”

Hypothesis:

- Mobile web applications can run on “any” browser enabled mobile device?

- Is this really true?
- Are all browsers actually equivalent?
- Do users get the same quality of user experience on all devices?
- Can the web actually offer the same quality of user experience as a native application?

No, it isn’t true. Browsers on phones are not equivalent to browsers on desktop devices.

Why use a Web App?

- Faster and cheaper development.
- It works on “any” browser.
- “No” download needed.
- Can use it to transition customers to native apps.
- You can’t exploit SEO with native apps – you can with web apps.
- Could use it as a form of prototype and for usability testing.
- Easier to get someone to view a link than it is to get them to download an app.

Native Apps

5 Pretty Good Reasons to Create Native Apps

1. Business Model
2. Feature Access
3. Connectivity
4. Local Storage
5. Interaction Control

Reason 1: Business Model

Hypothesis:

Easier for developers to charge for native applications.

- App Stores provide all infrastructure for collecting payments
- Developer can focus on development, not distribution

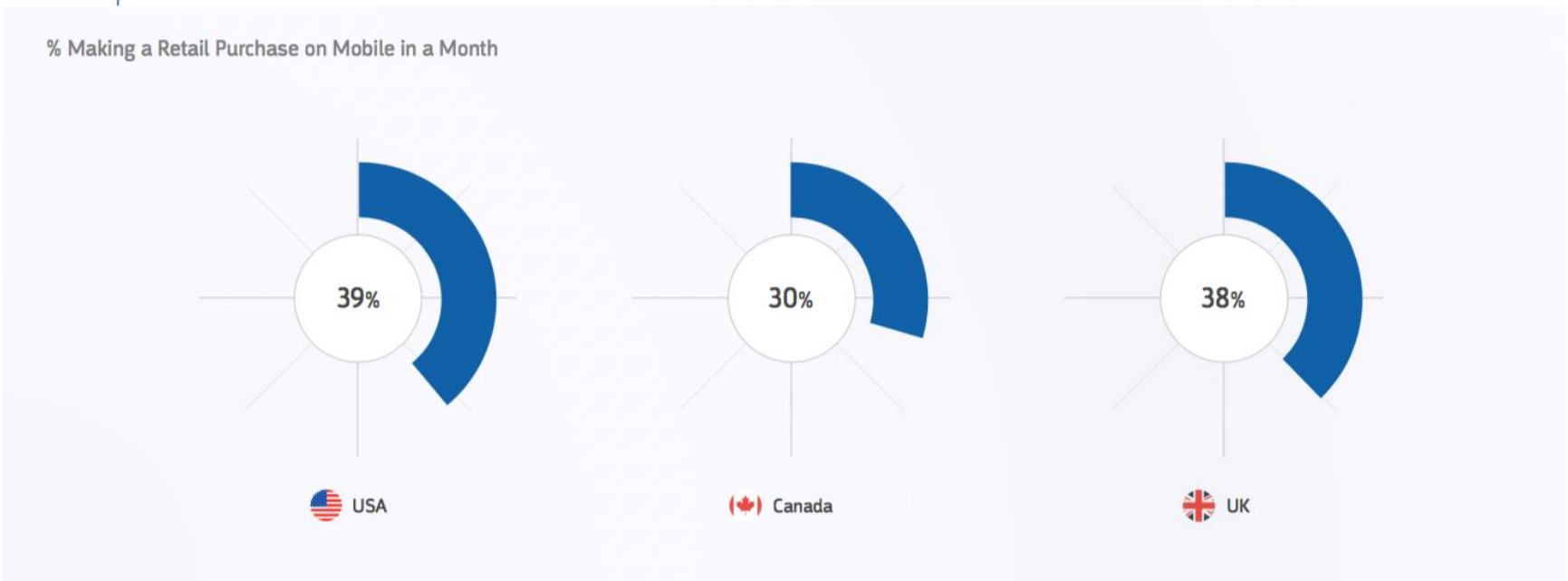
- But ... App Stores takes a cut.
  - Apple takes 30%
  - Same with Google Play Store (30%)

Also consider user viewpoint:

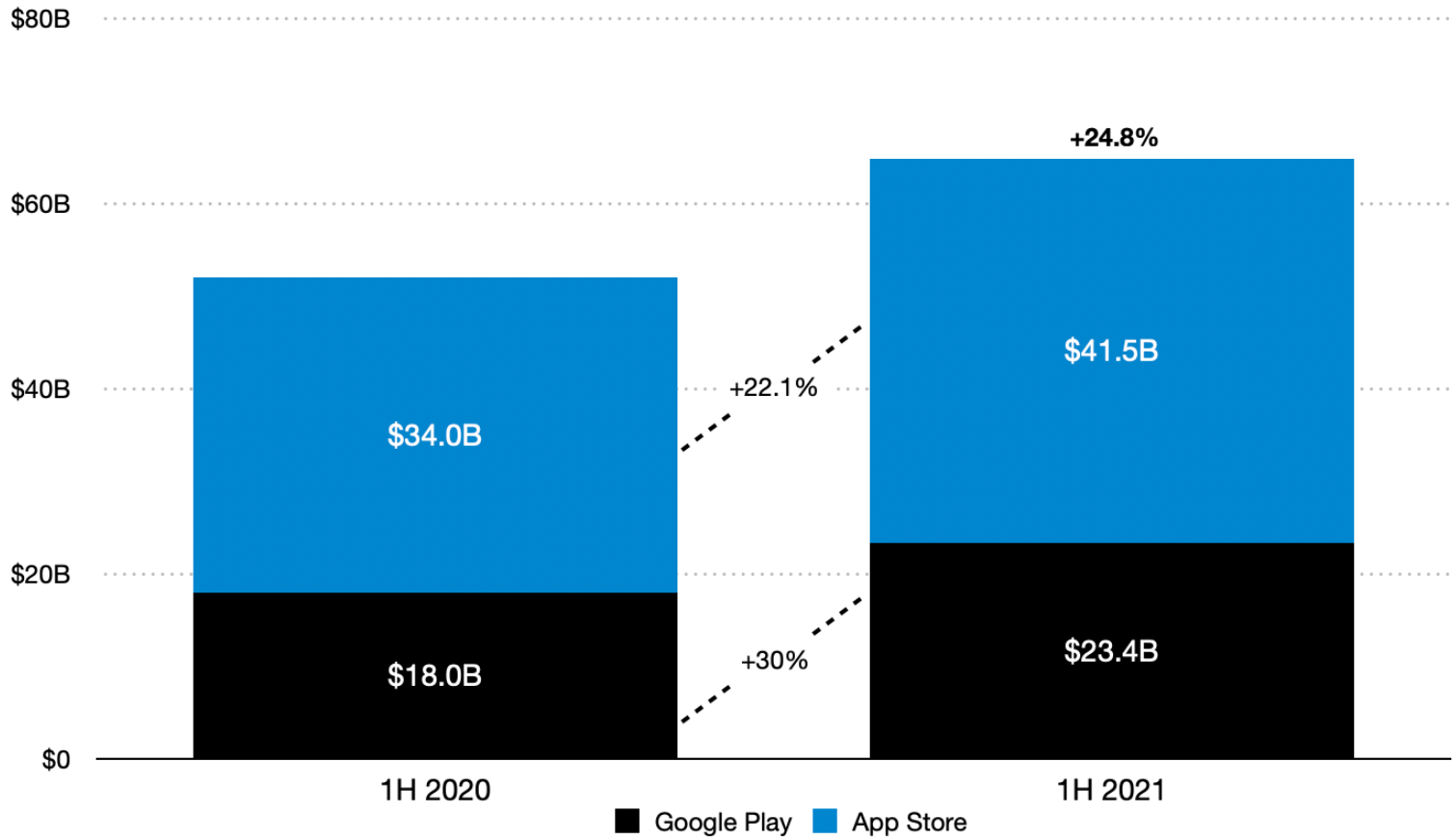
- Software as a product (pay to buy) v. Software as a service (pay/subscribe to use)
- Software as a service is typically less attractive to users

Fewer than 4 out of 10 mobile users purchase in a month

Under half of all mobile users in three major markets are making all mobile retail transactions



Global Consumer Spending in Mobile Apps and Games for First-Half 2021



Estimated consumer spending between January 1, 2020 and June 30, 2021

Source: Sensor Tower Store Intelligence

Reason 2: Feature Access

Hypothesis:

Native applications have better access to many device features and functions.

- Accelerometers
- Cameras
- Direct screen access
- Geo-location (GPS)
- Networking (Bluetooth, SMS, etc.)
- Local storage

- Using these features goes a long way to improving user experience.

"***Most of these features cannot be accessed via a browser…***"

Feature Access

WebAPI

“**WebAPI** is a term used to refer to a suite of device compatibility and access APIs that allow Web apps and content to access device hardware (such as battery status or the device vibration hardware), as well as access to data stored on the device (such as the calendar or contacts list). By adding these APIs, we hope to expand what the Web can do today to also include what only proprietary platforms were able to do in the past.”

You can view the full list here: https://developer.mozilla.org/en-US/docs/WebAPI.

MDN MOZILLA DEVELOPER NETWORK

Hardware access APIs

Ambient Light Sensor API

Provides access to the ambient light sensor, which lets your app detect the ambient light level in the vicinity of the device.

Battery Status API

Provides information about the battery's charge level and whether or not the device is currently plugged in and charging.

Geolocation API

Provides information about the device's physical location.

Pointer Lock API

Lets apps lock access to the mouse and gain access to movement deltas rather than absolute coordinates; this is great for gaming.

Proximity API

Lets you detect proximity of the device to a nearby object, such as the user's face.

Device Orientation API

Provides notifications when the device's orientation changes.

MDN MOZILLA DEVELOPER NETWORK

Other APIs

Alarm API

Lets apps schedule notifications. Also provides support for automatically launching an app at a specific time.

Simple Push API

Lets the platform send notification messages to specific applications.

Push API

Gives web applications the ability to receive messages pushed to them from a server, whether or not the web app is in the foreground, or even currently loaded, on a user agent.

Web Notifications

Lets applications send notifications displayed at the system level.

Any potential issues?

WebAPI has an extensive library to access a mobile device's hardware and data; however, many of the API libraries currently have **poor mobile browser support and compatibility**.

Example: Alarm API Browser Support

Browser compatibility

For obvious reasons, support is primarily expected on mobile browsers.

Desktop | **Mobile**

Feature	Android	Firefox Mobile (Gecko)	Firefox OS (Gecko)	IE Mobile	Opera Mobile	Safari Mobile
Basic support	No support	12.0 (12.0)	1.0.1	No support	No support	No support

However: If it can be done for Mozilla it can be done for any browser -- Provided you are willing to do some very intense low-level coding.

Feature Access

- Newer device features are only accessible through native applications or have restricted browser access:
  - Payment services
  - Augmented reality APIs

Feature Access - Summary

Native applications currently have better feature access than web applications …

- Feature access for web applications is improving quickly
- Any differences are slowly decreasing
- New features will always lag in getting web support

If you need strong feature access native will be a better choice.

Reason 3: Connectivity

Hypothesis:

Native applications run (locally) even with no connectivity.

- Do not use bandwidth after an initial purchase (or download).
- Give the user the freedom to use the application anywhere.
- Does not depend on a server being up somewhere.

But is this really true?

Connectivity

Thoughts:

- A lot depends on the application; Twitter and Skype are pretty useless without a connection, whereas Solitaire is highly usable without a connection.
- Is a smartphone ever really “disconnected?” (Yes, but it’s rare)
- Can one minimise the impact of the connectivity? (e.g., caching data when wifi is available, using SMS to avoid cellular data.)
- Can some functionality be made connection independent to allow the user to use the app in a limited fashion when disconnected (e.g., editing documents offline)?

Connectivity is highly influenced by many design decisions.

Native apps can often be made to work offline whereas web apps cannot.

Reason 4: Local Access

Hypothesis:

Native applications can store and access data stored locally, using a variety of methods: databases, files, etc.

- Allows a rich interaction between a variety of applications.
- Allows users to work and save their work on the mobile device from anywhere at any time.
- Allows access to local data, e.g., address book, photos, music.

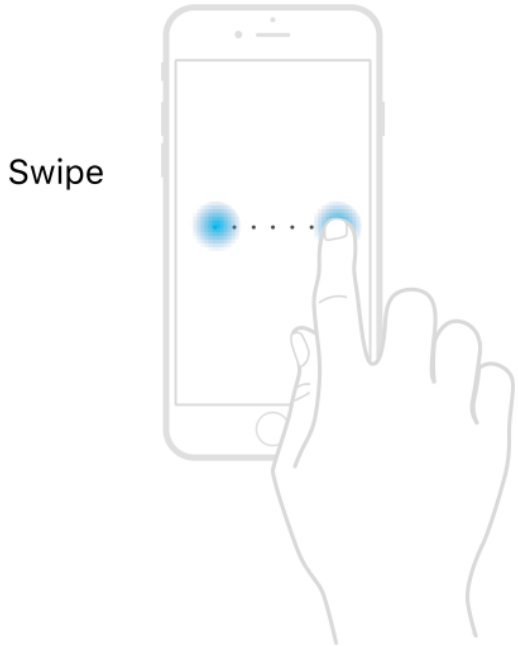
A very good reason for creating a native application!

Reason 5: Interaction Control

Hypothesis:

Allows developers to have fine-grained control over the screen, controls, and interactions.

- Not constrained by the browser
- Lower overhead
- Better response time (important for some games)



Native applications facilitate a high-quality user experience. ☺

Summary: Mobile Web or Native?
-- You should consider …

- Can/should you develop iOS and Android?
- Web Leverage of existing resources.
- Using the web as a prototype or entry version.
- Control over platform (i.e., independence from Google's or Apple's choices).
- Consumer expectations and familiarity with browsers and applications.
- Choice of business model.
- Feature access requirements.
- Usability when offline/disconnected.
- Availability and need for local storage.
- What kind of user experience are you trying to create?
- How much ubiquity do you need across devices?

Summary: What you should know …

- Given a scenario, know how to justify **why** you would build a **web app** or a **native application** specifically for a mobile device.
- Know the benefits of developing a native mobile application (v. a web application):
  - Users spend more time on mobile application than the browser.
  - Developers can leverage performance and hardware/OS features.
  - Major vendors, Google & Apple, benefit from owning and operating the app stores thus they will always support native developers (e.g. Apple Developer Account resources vs. an average web framework)
- Know the different business models for app distribution: **free**, **freemium**, **subscription**, and **paid**.