Hybrid Mobile Application Analysis and Guidelines

To blend the best of native and mobile Web browser capabilities, developers need to rate technical requirements and impediments in the context of business objectives.

Executive Summary

Along with cloud, social networking and advanced analytics, enterprise mobility is among the most important business-technology trends of our time. Together, the four technologies form the SMAC Stack™, and when holistically applied they can open new ways of working that enhance productivity and performance. In fact, many B2B organizations are adopting a “mobile first” mindset to better serve their constituents inside the four walls and with business partners and customers.

Enterprise mobile applications (from here on referred as mobile applications) are divided into three categories: native applications, mobile Web applications and hybrid applications. Much debate and deliberation is ongoing over the dominance of one approach over the others. Increasingly, however, enterprises are sidestepping those decisions and opting for whatever makes sense in each individual circumstance. Enterprises are quickly realizing that there is no “one-size-fits-all” solution to their mobile development process as the choice is dependent on business needs, app requirements, developer skill, development timelines and other factors.

- **Native applications are specifically developed for one platform and can take full advantage of device capabilities.** These are developed in the programming language that is directly supported by the platform. For example: the Facebook iOS application and Facebook Android app are both native applications, specifically developed for each platform.

- **Web apps are not exactly mobile apps but are Web sites that are mobile optimized.** Web apps resemble native applications, but ascended in popularity with the arrival of HTML5, which developers quickly realized could enable them to obtain native-like functionality in a browser-based environment. Today, as more and more sites use HTML5, the distinction between Web apps and regular Web pages has blurred.

- **A hybrid application is a mix of both native and Web technologies that are leveraged to deliver a mix of Web content and native capabilities.** Hybrid applications are sometimes extensions of Web applications wherein HTML content is rendered through a Webkit browser that is embedded within the native application. Hybrid applications are best suited for cross-platform requirements; that is, when the same HTML content needs to be accessed from different mobile platforms such as iOS, Android, Windows, etc. However, native features such as interaction with device features (i.e., camera, GPS, gestures, notifications, etc.) cannot be ignored. Hence, a hybrid approach takes into consideration both native and Web capabilities.
to provide an optimum cost-effective solution for cross-platform compatibility.

This white paper highlights criteria for selecting a mobile app implementation approach based on parameters such as usability, cost, time-to-market, development/support, etc. (whether hybrid or native). It also focuses on functional advantages and disadvantages, challenges, implementation methods and the impact on operational aspects such as maintenance for a hybrid application versus a native application.

Lastly, this paper is the outcome of our learning while developing a hybrid mobile application for a private banking institution. In that engagement, we evaluated and rated parameters such as effort, cost, security and performance – attributes that inform the mobile app evaluation framework discussed in this paper. This analysis will provide much-needed insight about factors to consider when defining an architecture or strategy for implementing a multi-platform, multi-device mobility solution, and can help organizations overcome the challenges they are likely to encounter.

In the context of this paper, it is assumed that a Web-based application already exists that can be adapted to the mobile channel.

**Enterprise Apps: An Anatomical Analysis**

Based on business imperatives, organizations can assess and then decide whether to choose a native mobile app implementation or following a hybrid approach is better suited to their needs. We suggest using an assessment approach similar to what is presented in Figure 1. For simplicity, the ranking is based on five stars, with five being the best fit score.

As Figure 1 reveals, if response time, user experience and offline support are important

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Native App</th>
<th>Hybrid App</th>
<th>Justification for the ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-platform Support</td>
<td>NA</td>
<td>*****</td>
<td>Hybrid apps can be ported to multiple platforms. However, for each platform, a specific native module/container must be developed. Support for older devices can be challenging for hybrid apps due to rich HTML features that might not work on devices with old browsers. However, a hybrid app can utilize device detection to trim features for low-end devices.</td>
</tr>
<tr>
<td>Response Time</td>
<td>*****</td>
<td>****</td>
<td>In terms of performance, native apps are faster. Typically, a native app uses back-end services (REST, Web services, etc.) to retrieve the data and represent it in the user interface (UI). Whereas in hybrid architecture, server-side components (presentation tier) return HTML markup which is rendered by a native browser packaged within the app. This causes additional overhead as compared with pure native applications.</td>
</tr>
<tr>
<td>User Experience</td>
<td>*****</td>
<td>***</td>
<td>Native apps are best in terms of usability, as they leverage device capabilities such as gestures as well as hardware controls such as a camera, GPS, accelerometer, compass, list of contacts, etc. In terms of look and feel, hybrid apps need extra effort to make them simulate a native app. It requires the adoption of third-party JavaScript JS libraries such as jQuery Mobile, scrolling libraries (e.g., swipe JS, iScroll, Zynga, etc.).</td>
</tr>
<tr>
<td>Development Cost</td>
<td>***</td>
<td>****</td>
<td>It’s arguably cheaper to develop hybrid apps, as the development process involves writing a native shell/container and wrapper over the existing presentation layer using third-party libraries. For native apps there are specialized skills required for each platform.</td>
</tr>
<tr>
<td>Support Cost</td>
<td>***</td>
<td>****</td>
<td>Maintaining a native app requires specialized skills (especially if they need access to multiple versions of the same information on different platforms). Changes have to be packaged in a new version and placed in the app/play stores. On the other hand, maintaining a hybrid app is as simple as maintaining a Web page, which can be conducted as needed.</td>
</tr>
</tbody>
</table>

Figure 1 continued on next page
## Finding the Best Fit App Dev Approach (continued from previous page)

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Native App</th>
<th>Hybrid App</th>
<th>Justification for the ratings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Reuse and Duplication</td>
<td>**</td>
<td>*****</td>
<td>Since platform-specific code needs to be written to build native apps, there is often a good deal of code duplication. Each platform-specific code needs to be rewritten in its OS supported language. A hybrid app can leverage common server-side components as the same HTML content is exposed for each platform.</td>
</tr>
<tr>
<td>Functional Testing</td>
<td>***</td>
<td>****</td>
<td>Native apps on multiple platforms must be tested and certified for each platform. Hybrid apps leverage server-side components which can be functionally certified for one platform that should work for all the platforms. However, UI/UX testing effort remains the same for both native/hybrid apps.</td>
</tr>
<tr>
<td>Off-line Support</td>
<td>*****</td>
<td>***</td>
<td>A native app is best if network connectivity isn’t required. In-browser caching is available in HTML5, but it’s still more limited than native.</td>
</tr>
<tr>
<td>Cost/Time-to-Market</td>
<td>***</td>
<td>*****</td>
<td>Rolling out a native app for multiple platforms is costly and time-consuming. Each platform-specific app must be tested and certified before it is rolled out to the market. In this sense, a hybrid app can better address multiple platforms and requires less time to develop and deploy.</td>
</tr>
<tr>
<td>Leveraging Existing Server-Side Components (Web Application)</td>
<td>**</td>
<td>****</td>
<td>In terms of leveraging existing server-side components, hybrid apps can utilize existing server-side resources. An existing Web application can be adapted for reuse to stream mobile-specific views for the hybrid app.</td>
</tr>
<tr>
<td>Resource Pool</td>
<td>**</td>
<td>*****</td>
<td>Web developers are more easily available as compared to mobile-platform-specific developers specialized in iOS or Android.</td>
</tr>
<tr>
<td>Usage Statistics/Analytics</td>
<td>***</td>
<td>****</td>
<td>While working with hybrid apps, existing Web/server-side applications’ usage statistics or usage logging can be utilized as is. No special engineering is required to collect data with respect to analytics. For native apps, collecting this data means additional storage space is required on devices and posting those back to the server would be an additional task. Usage stat data archival or purging too would be additional work in the case of pure native applications.</td>
</tr>
<tr>
<td>Support for Multiple OS &amp; Form Factors</td>
<td>*</td>
<td>****</td>
<td>Developing a pure native app for business functionality and rolling it out for multiple form factors and multiple OS platforms would mean enterprises end up developing a minimum of four applications (iOS, Android, mobiles and tablets). Whereas a single responsive Web application and native shell can serve as a hybrid application for a multi-device platform. However, an RWD-based application has its own challenges, and thorough analysis is required before the start of the project.</td>
</tr>
<tr>
<td>Future Release/Enhancements</td>
<td>*</td>
<td>*****</td>
<td>For a native app, improvising an existing feature or implementing a new one means a rollout through a platform-specific distribution store is required. Also, users who do not update the app must be supported, and they will be restricted from using new features. In the case of security patches, this is a severe threat to the user and his data. With a hybrid application, rolling out new features or enhancements is much easier as no client app distribution is required.</td>
</tr>
<tr>
<td>Security</td>
<td>**</td>
<td>****</td>
<td>Typically, enterprise applications are exposed over the Web and consumed via hybrid applications using an https protocol, which means that no sensitive data is stored on a device. It reduces the risks of data theft or an application being exposed to potential hackers and malware in users’ devices.</td>
</tr>
</tbody>
</table>

![Figure 1](image-url)
business requirements and are ranked higher than parameters such as time-to-market, code reuse and cost of development and support, then native implementation should be the preferred option. However if multi-platform support, time-to-market, leveraging existing server-side components and overall cost of development and long-term support are important, then a hybrid app implementation is better.

In the next section we will discuss hybrid application architecture and implementation challenges.

Figure 2 depicts a high level logical architecture of a typical hybrid app. Here, device detection and the Webview (an embedded Webkit browser) component on the mobile device are the most important components.

The Webview or the embedded Webkit browser can be programatically used to invoke an HTTP URL and fetch the content from the server. In Android and iOS apps, it is also possible to call the JavaScript functions on the HTML page rendered on the browser via the native code, and vice versa.

Challenges, Work-arounds in Hybrid App Implementation

There are numerous challenges with respect to building a mobile hybrid application. In this section we will elaborate on a few of these issues and offer ways to circumvent them.

Usability and UX Capabilities

Some of the challenges with respect to rolling out Web UIs as mobile applications are related to the following UX capabilities, which have limited support in the traditional HTML world.

- Page transitions like pull up, pull down, pop, fade, flip features and other animations.
- Swipe-in, swipe-out gestures.
- Charting controls.
- Ability to show different content/UI depending on screen orientation.

Device Form-Factor Adaptation

Device diversity and multi-platform devices reflect today’s mobile device world where many devices come with varying screen sizes. Server-side pages need to address different devices and

Hybrid App Architecture

![Hybrid App Architecture Diagram]

Figure 2
form factors; this becomes quite challenging. To meet the requirement for streamlined content from the server for specific device form factors, device detection and adaptive/responsive Web design principles are helpful when designing the application.

**Device Detection**

To perform device detection on the server side, consider the following:

- **Device detection libraries**: Libraries such as WURFL\(^8\) and DeviceAtlas\(^9\) can be leveraged to retrieve device information on the server side.

- **Custom device detection on server side**: Alternatively, custom device detection code can be written on the server side to parse the device information that can be passed as part of the user/agent string that moves from the client (device) in each request to the server. The device form factor can be extracted from the user/agent string and stored on the server side for later use.

Our recommendation is to use a custom, homegrown device detection if the targeted device and platforms are small.

**Adaptive/Responsive Web Design**

Using the concepts of adaptive design, device-detection-specific streamlined content that is optimized for the mobile device can be rendered from the server based on the device’s form factor. The advantage of adaptive Web pages specifically for mobile devices is that streamlined content is sent from the server, which conserves bandwidth. With adaptive content delivery, the server hosting the Web site detects the requesting devices, and uses this information to deliver different sets of HTML and cascading style sheet code based on the characteristics of the devices that have been detected.

Adaptive Web design also encompasses a range of other strategies that, when combined with responsive design techniques, enable app developers to deliver the best possible user experience to the widest possible audience. The responsive design aspect of adaptive design involves the implementation of various design factors such as flexible layouts, CSS file alternatives and flexible images activated using media queries. Having a breakpoint within the styles, at a minimum width of 600px, allows developers to change the look and feel of the app depending on the orientation of the device, while addressing the display requirements of a wide range of devices. This means that numerous functionalities and environmental factors can be handled in the most intuitive way, depending on the particular device used to access the Web site.

The main benefits of using adaptive/responsive Web design include:

- **Full content adaptation**: Markup optimization.
- **Performance**: Optimized markup, JS.
- **Low-end device targeting**: Enabled by partial support for HTML standards.
- **Future ready**: The same solution can be extended for smart TV and other new devices that support browser access.

---

### Feature | Description | HTML(Web) Alternative
---|---|---
**Swipes** | Capability to support swipes/gestures. | Use JS libraries like swipe JS and Jquery mobile. It provides capability to support gestures/swipes in a hybrid solution.

**Pull-Up/-Down** | Capability to pull-up/down to load more data from the back end. | Use JS libraries like iScroll or Zynga.

**Charting Controls** | Capability to show interactive graphs/charts. | Use JS libraries like flotr\(^2\) or RGraph.\(^6\) For J2EE enterprise applications, jFreeChart\(^7\) is a feasible solution. ASP.NET provides chart controls for a similar purpose.

**Different Content/UI on Screen Orientation** | The business requirement may be to show different or more content in a landscape view compared with a portrait view. | This can be achieved by using CSS3 media queries to show different content based on device width/height.

Figure 3
Respecting Device Diversity

Another key challenge for mobile apps is to address and support a diverse array of devices. Numerous devices have emerged with different form factors, platforms and platform-specific OS, hence it is increasingly difficult to build one app for both native and hybrid use. Features such as pull to refresh and swipe are based on browser capabilities, and many legacy devices do not support modern browsers (such as Safari and Chrome) and hence have limited support for HTML5, advanced JSJS and CSS3 capabilities.

In our engagements, the following device-specific challenges have surfaced:

- **Android Samsung’s S3 and S4 numeric keyboard doesn’t have a decimal key for html input field with type as number.** An alternate solution was to go for an HTML keyboard based (JavaScript based) approach.
  - A possible solution, which we implemented, was to provide an HTML-based keyboard in relevant screens for entering the amount value with a decimal point.

- **Android devices with OS version 2.3 have issues with JavaScript.**
  - One-click functions are not invoked under specific scenarios (observed on Samsung S2 OS version 2.3.x).

- **iOS 7 UI WebView component starts its rendering after the navigation bar.** This makes the HTML markup (typically the header area) look inappropriate.
  - Via device detection, if request is from iOS7 device, then add a custom CSS which will margin top to HTML body element.

- **On iPhone 4, iOS 6 devices UI Web View is truncated at the bottom by roughly 10 px.** This causes the HTML (typically the footer area) to look truncated.
  - Again, detect the request for iPhone4, iOS 6 using device detection and add a custom CSS to add padding-bottom to the first element in your HTML body.

In order to support different devices with limited capabilities, special handling and implementation checks need to be added in code. However, this makes the application complex and difficult to maintain in the long run. So, there is a trade-off between maintainability vs. market reach/rollout.

Native/Web Communication

Communicating from the native OS-specific code to Web parts and vice versa is another typical technical requirement in hybrid architecture. Here are some of the use cases we have run into wherein native/Web communication is needed.

- On a Web header, there is a button to click that opens a native sliding menu.
- When the user pushes the hardware back button, the native app should take user back to the previous Web page.
- When clicking on a Web page needs to enable hardware controls like camera/flashlight.

The native/Web communication can be achieved using a JavaScript bridge in the case of Android or writing a custom native interceptor that intercepts every Web call to the server and custom implementation which handles those server calls based on specific URL nomenclature.

In both Android and iOS, it is also possible to invoke the JavaScript function on the rendered HTML page from the native code.

Testing

App testing is another challenging aspect of the development lifecycle. Regarding end-to-end testing of mobile apps, developers should think holistically about:

- Functional testing.
- Device testing.
- Load and performance testing.
- Testing of mobile-specific scenarios.

The functional testing of a hybrid app where Web parts are involved can be carried out on modern browsers such as Chrome/Firefox. Both Chrome and Firefox come with additional plug-ins that can be used to create custom user-agent headers that mimic the mobile user-agent. Browsers can also be used to test Web pages for different screen sizes and form factors. This way, functional testing can be achieved easily on the browser and to a large extent can be automated by using tools such as QTP or Selenium.8

Device testing is exceedingly complex for mobile apps. The reason: beyond functional testing, performance must be assessed on the individual device level as well as from a usability and UX
perspective. To test usability and UX capability on an individual device level, an offline version of the app should be prepared that works with bundled static HTML pages.

Another issue for hybrid apps: load and performance tests where Web parts are involved can be easily carried out on the browser or by using a tool like LoadRunner.

It is recommended that enterprise mobile apps be put through a detailed round of verification by mobile testing experts. Scenarios specific to mobile architecture that should be tested thoroughly include:

- Application put to background while in use.
- Tracking inactivity/idle time-out.
- Intermittents to an ongoing application such as SMS notification, incoming call.
- OS running low on memory.
- Frequent switches between views that load large amounts of data (typically graphical resources like high-res images).
- Features that use native hardware capabilities such as camera, flash, vibrator, etc.

One suggested approach is to test the application through a tool like MonkeyRunner which can be used for test automation for regression, functional testing and running a suite of tests across multiple devices.

Looking Forward

The choice of whether to take a hybrid or native approach should be determined by the assessment your organization conducts using the framework highlighted in Figure 1. It should prioritize long-term cost, usability and multi-platform support to make sure the best performing and appropriate apps are built for the business requirement/objective at hand.

Glossary of Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2C</td>
<td>Business to consumers.</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascaded style sheet.</td>
</tr>
<tr>
<td>RWD</td>
<td>Responsive Web design.</td>
</tr>
<tr>
<td>UX</td>
<td>User experience.</td>
</tr>
<tr>
<td>Swipe JS</td>
<td>A JS library for swipe features.</td>
</tr>
<tr>
<td>JQuery Mobile</td>
<td>A JS library for mobile browser.</td>
</tr>
<tr>
<td>iScroll</td>
<td>A JS library for scrolling.</td>
</tr>
<tr>
<td>Zynga Scroll</td>
<td>A JS library for scrolling.</td>
</tr>
</tbody>
</table>

Footnotes

About the Authors

Amit Tank is a Solutions Architect within Cognizant’s Banking and Financial Services Business Unit. He has over 12 years of industry experience across several sectors including (but not limited to) software product development, professional services, research and development, manufacturing engineering and software applications. Amit has architected, designed and developed critical business-centric enterprise applications for companies in the insurance, mortgage, banking and financial services industries. He earned his bachelor’s degree in engineering from NIT Durgapur, India and is a TOGAF 9 certified enterprise architect. Amit can be reached at Amit.Tank@cognizant.com.

Chintan Desai is a Project Manager within Cognizant’s Banking and Financial Services Business Unit. He has 10-plus years of experience leading all phases of diverse technology projects, and over seven years of computer programming experience using C and Java in projects involving mobility (Android), enterprise content management and portals. He received a bachelor of engineering degree in computer science from DDIT, Nadiad. Chintan can be reached at Chintan.Desai@cognizant.com.