Mobile and IBM Worklight Best Practices

WebSphere User Group, Edinburgh, 24th Sept 2013

Andrew Ferrier
andrew.ferrier@uk.ibm.com

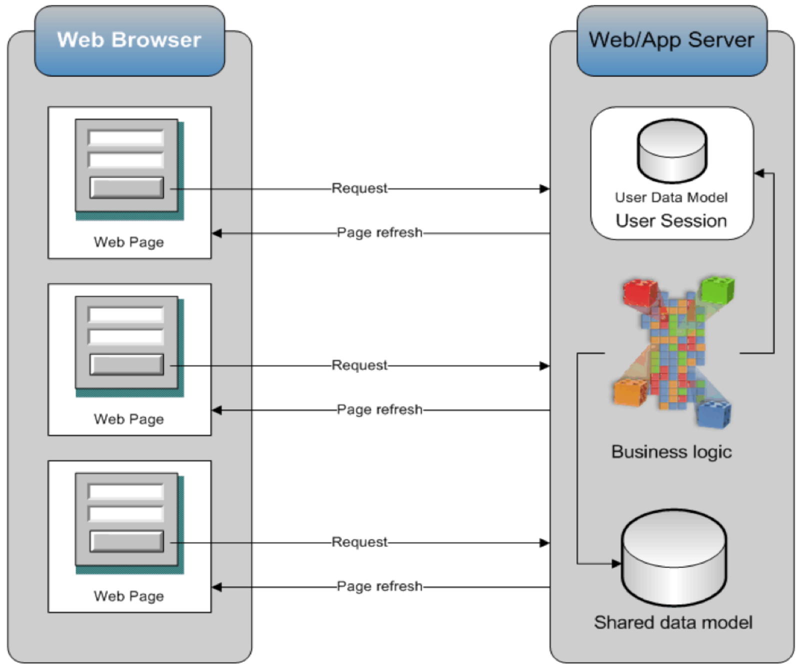
Contributions from:
Nick Maynard
Sean Bedford
Jon Marshall
and others….

Agenda

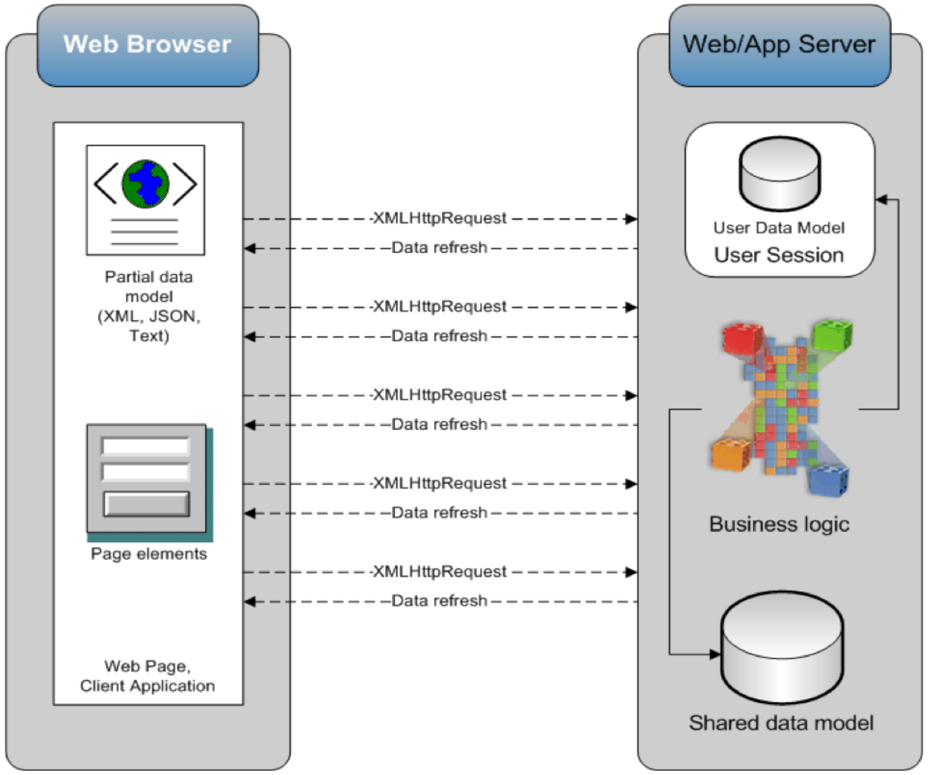
- **Recap**: Web 2.0 & Mobile Landscape
- **Development Time**
  - Toolkits & Frameworks
- **“Run” Time**
  - RESTful Services and WL Adapters
  - Worklight Lifecycle – Build, Test, Deploy
- **Updates/Other Best Practices**

Web 1.0: what we used to do

- Static HTML content, little-to-no-dynamicity
- Server-side-driven content
- Perhaps with a small amount of JavaScript for effects or form validation
- Traditionally written with a variety of technologies – Servlets, JSPs, PHP, etc.



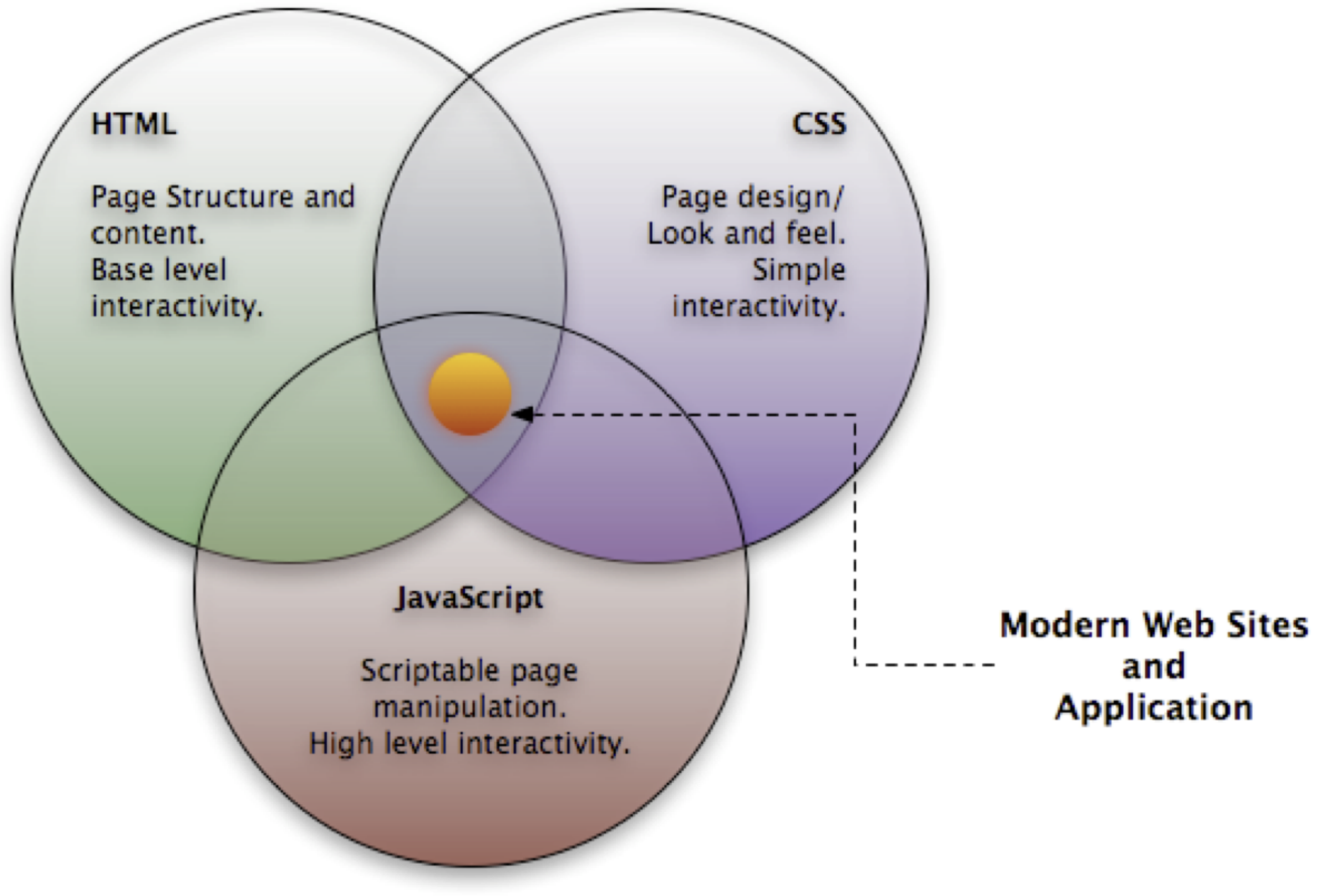
Embrace Web 2.0



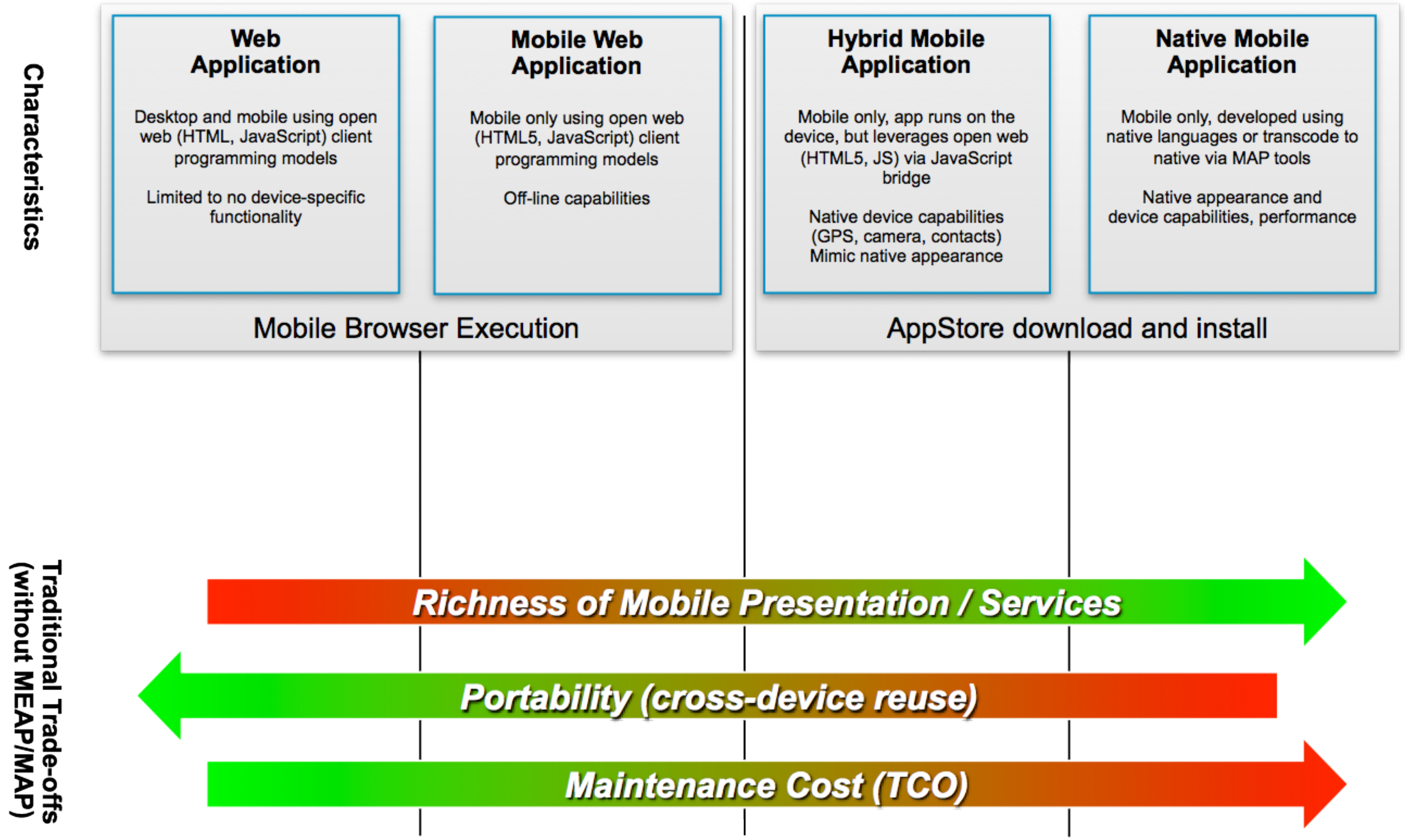
- Rich client-side JavaScript
- XHRs for data over RESTful Services
- JSON Payloads

- ~~Server-side-driven content~~
- ~~J2EE, Servlets, JSPs, PHP, etc.~~

Hire the Skills Needed! (The Programming Model)

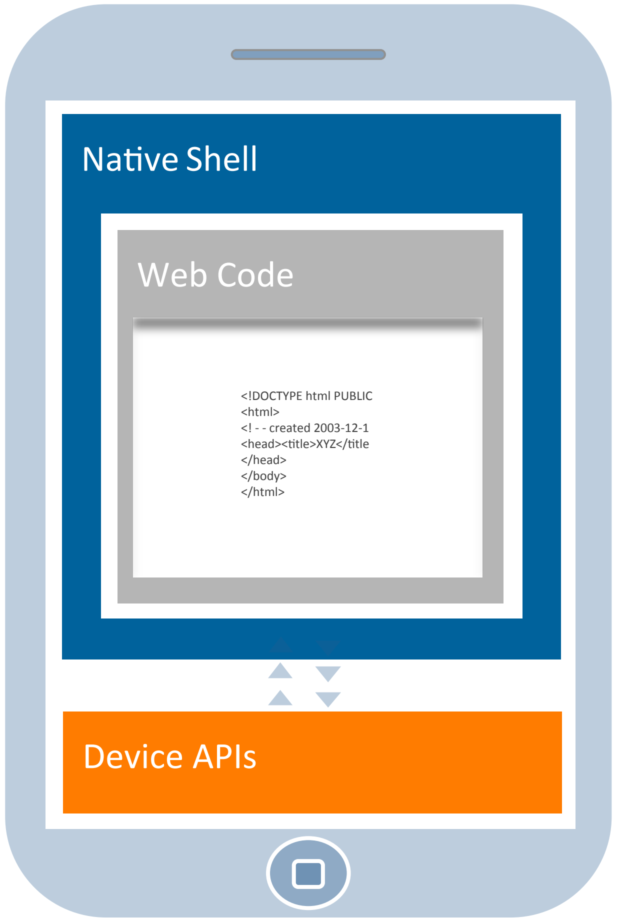

HTML
Page Structure and content. Base level interactivity.
CSS
Page design/ Look and feel. Simple interactivity.
JavaScript
Scriptable page manipulation. High level interactivity.
Modern Web Sites and Application

Understand the Mobile Landscape



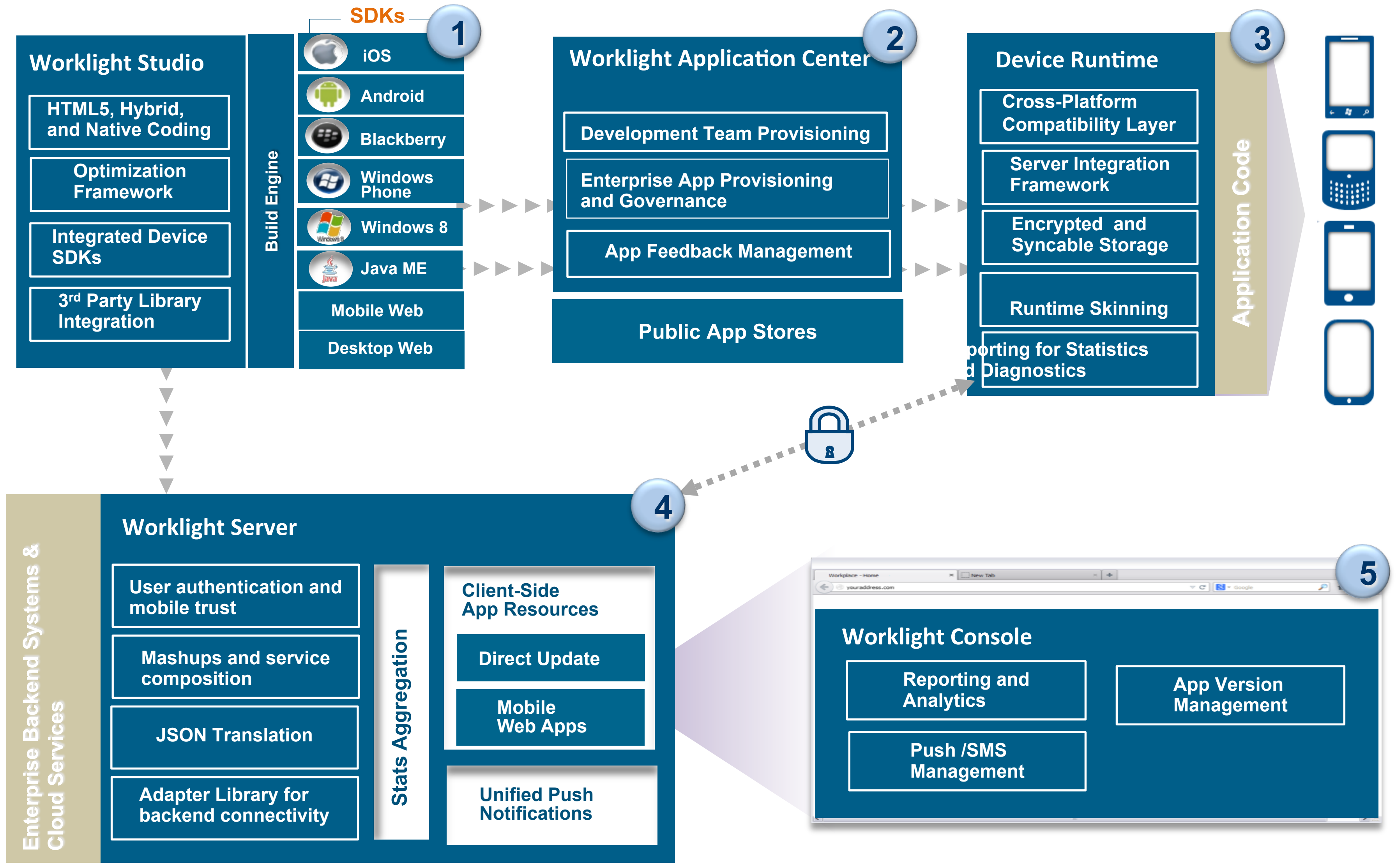
Characteristics
Web Application
Desktop and mobile using open web (HTML, JavaScript) client programming models
Limited to no device-specific functionality
Mobile Web Application
Mobile only using open web (HTML5, JavaScript) client programming models
Off-line capabilities
Hybrid Mobile Application
Mobile only, app runs on the device, but leverages open web (HTML5, JS) via JavaScript bridge
Native device capabilities (GPS, camera, contacts) Mimic native appearance
Native Mobile Application
Mobile only, developed using native languages or transcode to native via MAP tools
Native appearance and device capabilities, performance
Mobile Browser Execution
AppStore download and install
Traditional Trade-offs (without MEAP/MAP)
Richness of Mobile Presentation / Services
Portability (cross-device reuse)
Maintenance Cost (TCO)

Understand Worklight



```
<!DOCTYPE html PUBLIC
<html>
<! - - created 2003-12-1
<head><title>XYZ</title
</head>
</body>
</html>
```

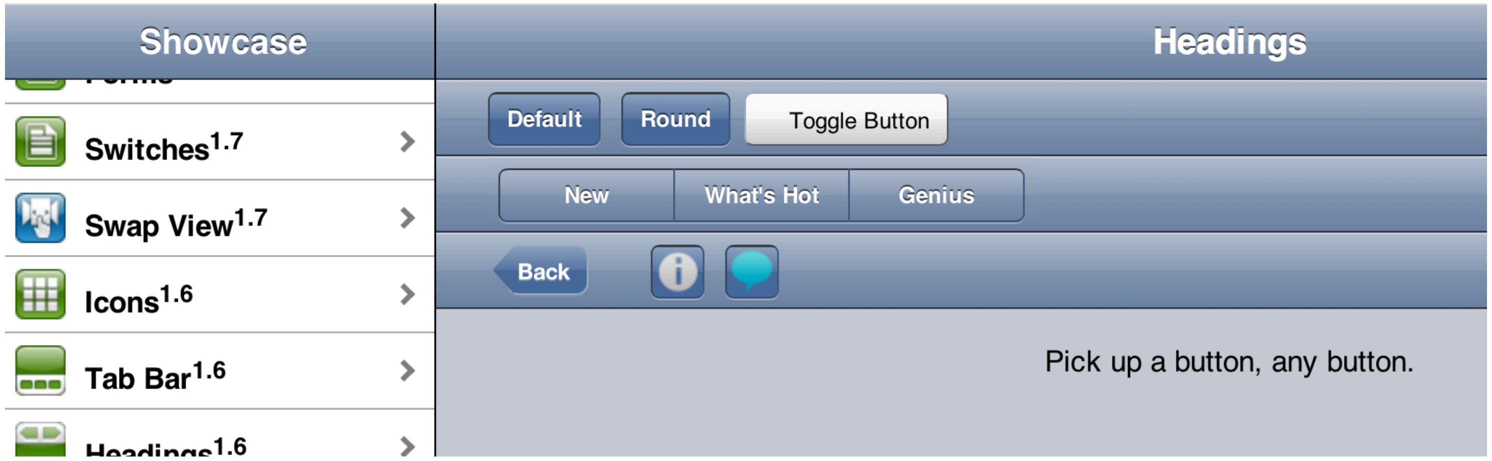
Understand Worklight


SDKs
1
Worklight Studio
HTML5, Hybrid, and Native Coding
Optimization Framework
Integrated Device SDKs
3rd Party Library Integration
Build Engine
iOS
Android
Blackberry
Windows Phone
Windows 8
Java ME
Mobile Web
Desktop Web
2
Worklight Application Center
Development Team Provisioning
Enterprise App Provisioning and Governance
App Feedback Management
Public App Stores
3
Device Runtime
Cross-Platform Compatibility Layer
Server Integration Framework
Encrypted and Syncable Storage
Runtime Skinning
porting for Statistics
d Diagnostics
Application Code
4
Worklight Server
Enterprise Backend Systems & Cloud Services
User authentication and mobile trust
Mashups and service composition
JSON Translation
Adapter Library for backend connectivity
Stats Aggregation
Client-Side App Resources
Direct Update
Mobile Web Apps
Unified Push Notifications
5
Worklight Console
Reporting and Analytics
App Version Management
Push /SMS Management

TOOLKITS AND FRAMEWORKS

Use a Toolkit

- *JavaScript-based libraries, written in JavaScript, used on top of JavaScript itself*
- Why?
  - Smooth out the JavaScript's rough edges
  - Add additional features, UI widgets, etc.



Use Dojo

- The largest players in the market are

jQuery write less, do more.   dōjō

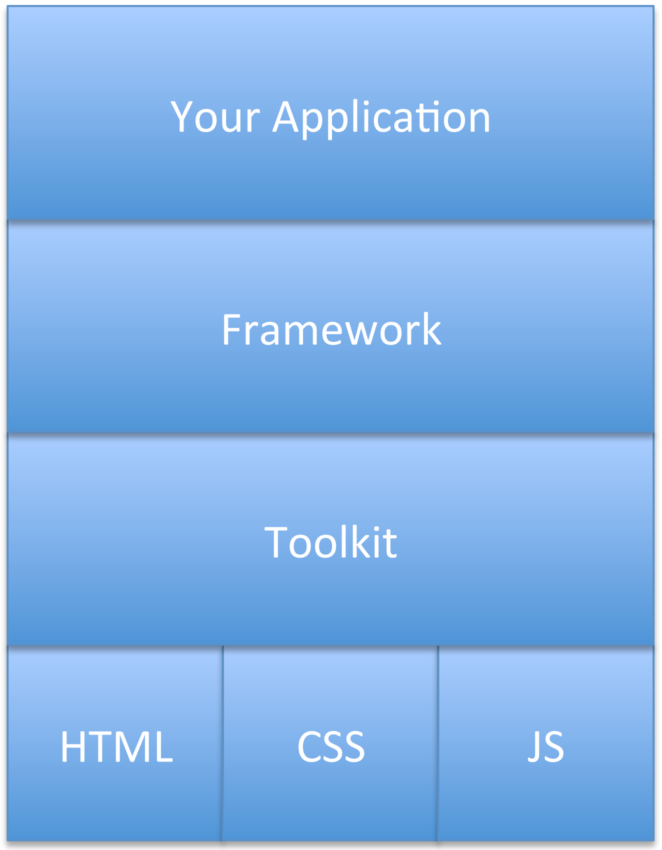
- Generally, IBM ‘prefers’ Dojo
  – Shipped with *IBM Worklight, WebSphere Feature Pack for Web 2.0 and Mobile*, etc..

Why Dojo?

- Enterprise-grade toolkit and feature set
- Stronger support for structuring large applications
  - e.g. Class system (`dojo/declare`)
- Better focus on internationalization, accessibility, etc.
- But **jQuery** is a supported choice too for Worklight and still a sensible choice

Consider using framework(s)

- Coding without a JS **toolkit** in 2013 is like entering the program in binary
- **Frameworks** sit on top of a toolkit, but gives you other things that are missing.



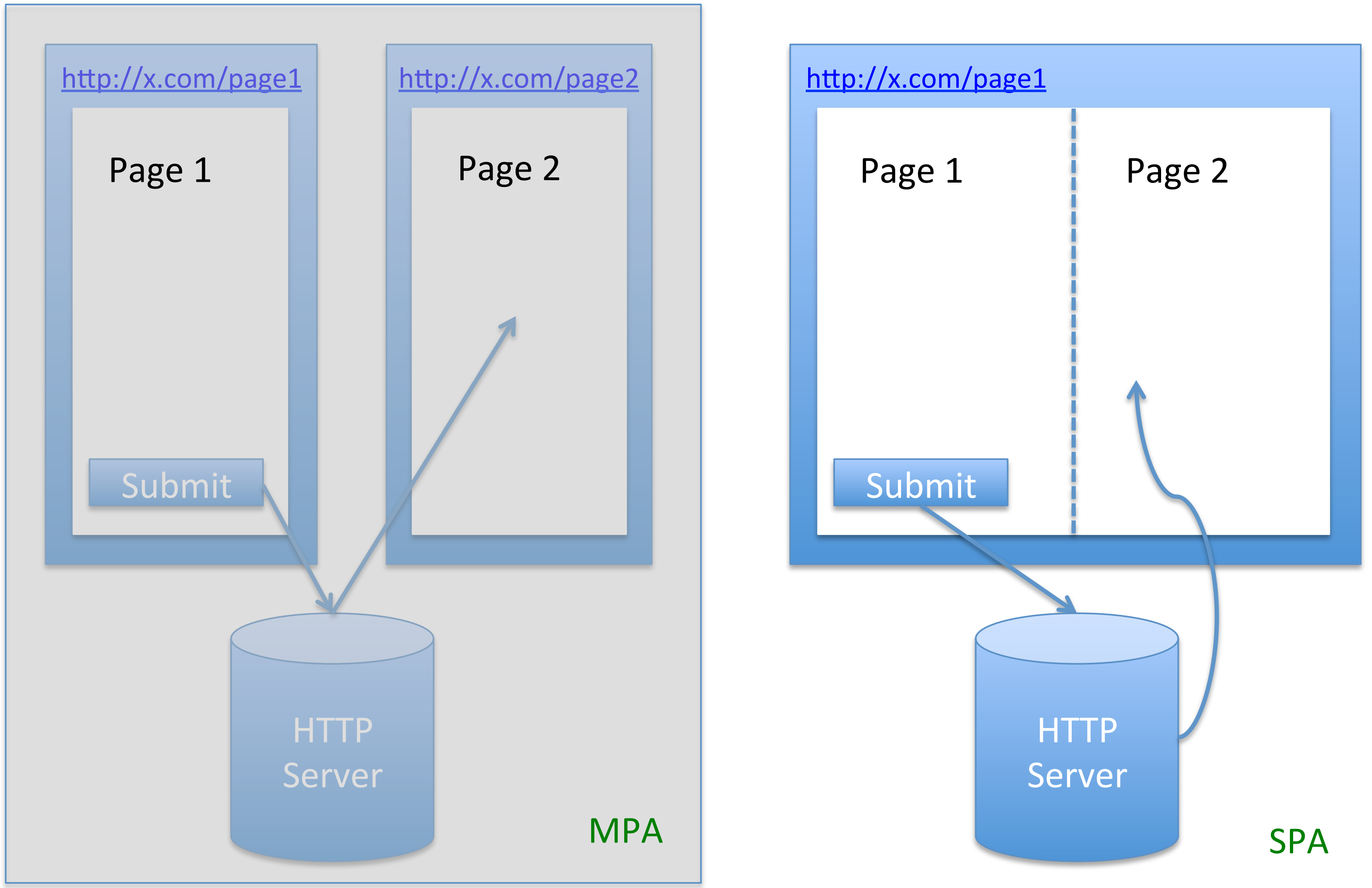
Consider using framework(s)

- For example, a framework might give you:
  - Endpoint management (stubbing)
  - State / session management
  - Responsive Design Benefits (e.g. dojox/app)
  - Templating
  - Single-page architecture support
  - Standardised error-handling
  - *(… other application-level stuff)*

Framework Options

- For Dojo:
  - Dojo itself - **dojox/mobile**, **dojox/app, dojox/mvc**
  - issw/mobile & issw/pocMobile
  - Your own custom framework
    - Not as bad an idea as it sounds!
- For jQuery:
  - Angular (MVW), mustache (templating), RequireJS (code loading), Knockout (MVC), Backbone (MVC), Handlebar (templating) etc...

Prefer Single Page Architecture

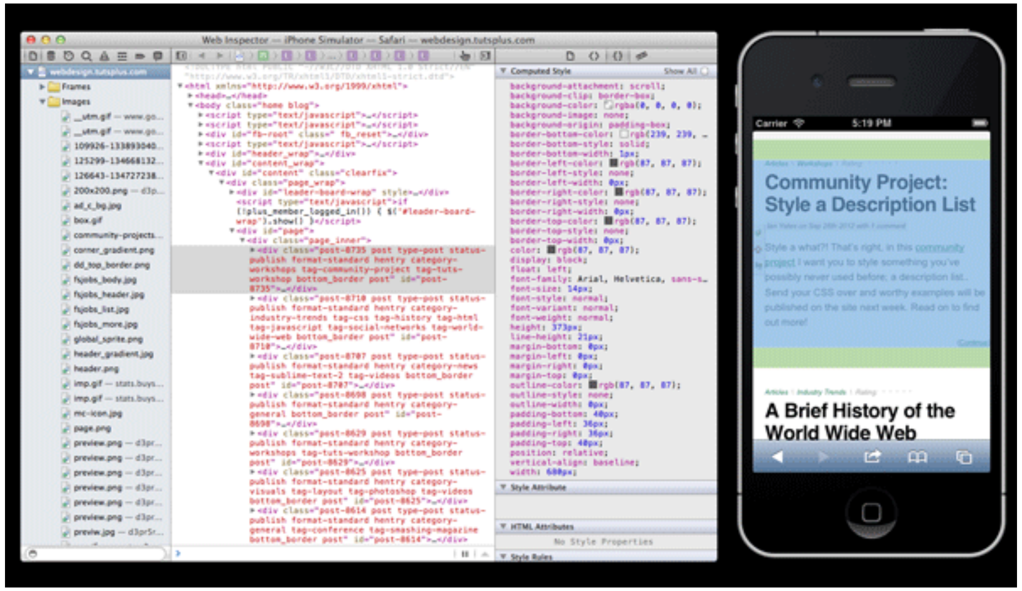


Prefer Single Page Architecture

- **(… for mobile at least)**
- Only one `.html` page
- Improves performance
- Dojo Mobile has this concept built in – `dojox/mobile/View`
- Reuse this concept for Hybrid too

Understand Debugging Options

- iOS 6+: Web Inspector (Physical & Emulated Phone)
- Android 4.x: Chrome Remote Debugging
- Desktop Browser with Debugging Support – Chrome, Firefox + Firebug (plain or Worklight simulator)
- Worklight logging
- Etc…



Web Inspector — iPhone Simulator — Safari — webdesign.tutsplus.com
webdesign.tutsplus.com
Frames
Images
Computed Style
Show All
Style Attribute
HTML Attributes
No Style Properties
Style Rules
Carrier
5:19 PM
Community Project: Style a Description List
A Brief History of the World Wide Web

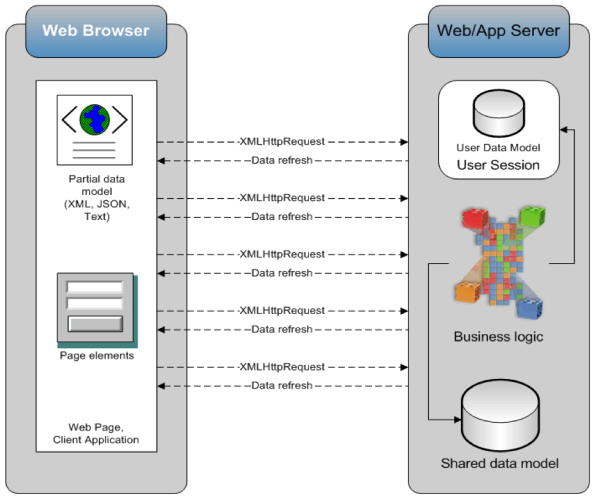
```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
▼<html xmlns="http://www.w3.org/1999/xhtml">
  ▶<head>…</head>
  ▼<body class="home blog">
    ▶<script type="text/javascript">…</script>
    ▶<script type="text/javascript">…</script>
    ▶<div id="fb-root" class=" fb_reset">…</div>
    ▶<script type="text/javascript">…</script>
    ▶<div id="header_wrap">…</div>
    ▼<div id="content_wrap">
      ▼<div id="content" class="clearfix">
        ▼<div class="page_wrap">
          ▶<div id="leader-board-wrap" style>…</div>
           <script type="text/javascript">if
           (!plus_member_logged_in()) { $('#leader-board-
           wrap').show() }</script>
          ▼<div id="page">
            ▼<div class="page_inner">
              ▶<div class="post-8735 post type-post status-
               publish format-standard hentry category-
               workshops tag-community-project tag-tuts-
               workshop bottom_border post" id="post-
               8735">…</div>
              ▶<div class="post-8710 post type-post status-
               publish format-standard hentry category-
               industry-trends tag-css tag-history tag-html
               tag-javascript tag-social-networks tag-world-
               wide-web bottom_border post" id="post-
               8710">…</div>
              ▶<div class="post-8707 post type-post status-
               publish format-standard hentry category-news
               tag-sublime-text-2 tag-videos bottom_border
               post" id="post-8707">…</div>
              ▶<div class="post-8698 post type-post status-
               publish format-standard hentry category-
               general bottom_border post" id="post-
               8698">…</div>
              ▶<div class="post-8629 post type-post status-
               publish format-standard hentry category-
               workshops tag-tuts-workshop bottom_border
               post" id="post-8629">…</div>
              ▶<div class="post-8625 post type-post status-
               publish format-standard hentry category-
               visuals tag-layout tag-photoshop tag-videos
               bottom_border post" id="post-8625">…</div>
              ▶<div class="post-8614 post type-post status-
               publish format-standard hentry category-
               general tag-conference tag-smashing-magazine
               bottom_border post" id="post-8614">…</div>
```

```
background-attachment: scroll;
background-clip: border-box;
background-color: rgba(0, 0, 0, 0);
background-image: none;
background-origin: padding-box;
border-bottom-color: rgb(239, 239, …
border-bottom-style: solid;
border-bottom-width: 1px;
border-left-color: rgb(87, 87, 87);
border-left-style: none;
border-left-width: 0px;
border-right-color: rgb(87, 87, 87);
border-right-style: none;
border-right-width: 0px;
border-top-color: rgb(87, 87, 87);
border-top-style: none;
border-top-width: 0px;
color: rgb(87, 87, 87);
display: block;
float: left;
font-family: Arial, Helvetica, sans-s…
font-size: 14px;
font-style: normal;
font-variant: normal;
font-weight: normal;
height: 373px;
line-height: 21px;
margin-bottom: 0px;
margin-left: 0px;
margin-right: 0px;
margin-top: 0px;
outline-color: rgb(87, 87, 87);
outline-style: none;
outline-width: 0px;
padding-bottom: 40px;
padding-left: 36px;
padding-right: 36px;
padding-top: 40px;
position: relative;
vertical-align: baseline;
width: 600px;
```

RESTFUL SERVICES AND WL ADAPTERS

RESTful Services

- The world (at least UIs) are moving to simpler services
  - A RESTful style - plain HTTP GET, PUT, POST, DELETE
  - JSON as the data format
- Practically *mandatory* for consumption by Web 2.0 clients



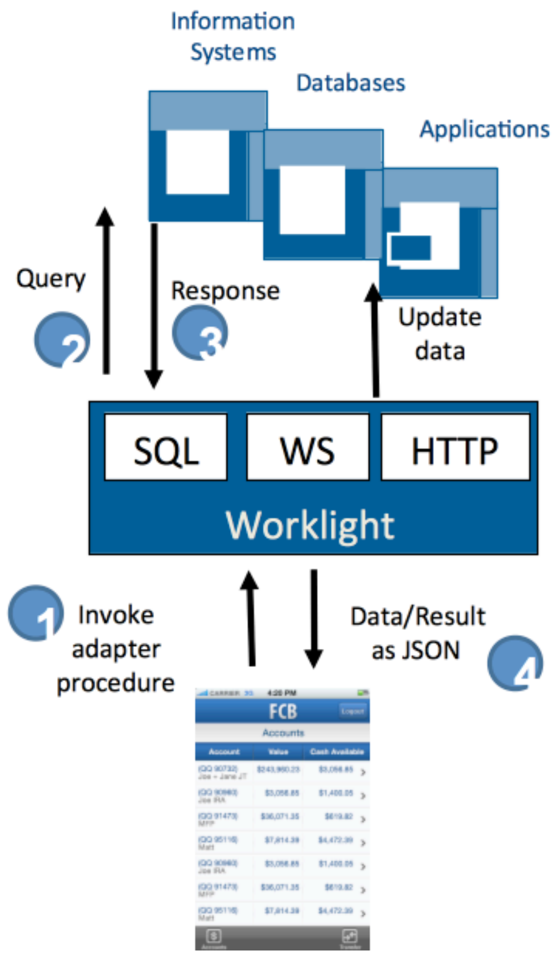
```
GET http://mycorp.com/
customer/1234

{
  “name”: “Fred Bloggs”,
  “address”: “123 Anytown”
}
```

RESTful Best Practices for Mobile Web

- Use verbs liberally: GET, PUT, POST, DELETE
  - http://mycorp.com/services/~~create~~Customer
- Keep them stateless (independent)
- Don’t send data that’s not needed
  - Keep payloads small
  - Combine related services
- Think about cacheability
- Think about pagination / querying / sorting

WL Adapters

- WL adds adapter framework
  - Server-side JS and XML components
  - Client-side invocation using JS API
- Supports HTTP, JMS, SQL, and Cast Iron adapter types
  - Most common use is HTTP adapter to integrate with JSON/REST or SOAP/HTTP



WL Adapters - REST & HTTP

- You could use RESTful services directly from WL container with conventional XHRs, but you lose:
  - **WL's authentication mechanism for services**
  - The ability to use the WL server as a "choke point"
  - WL Logging/Auditing
  - Analytics integration – Tealeaf usage is easier

Re-expose even RESTful services

- Even for services already exposed over REST, re-expose them using the WL HTTP Adapter.
  - This is comparatively straightforward to do.
- You can also use SOAP services from WL
  - Abilities are limited at the moment so for more sophisticated scenarios, consider an ESB (e.g. Cast Iron)

Consider Service & Adapter Versioning

- For RESTful Services:

  URL: /maps/version/2/map?...

  Custom HTTP Header: X-Version 2.1

  Media Types/Content Negotiation: application/json;version=1

- Versioning Worklight Adapters requires renaming them

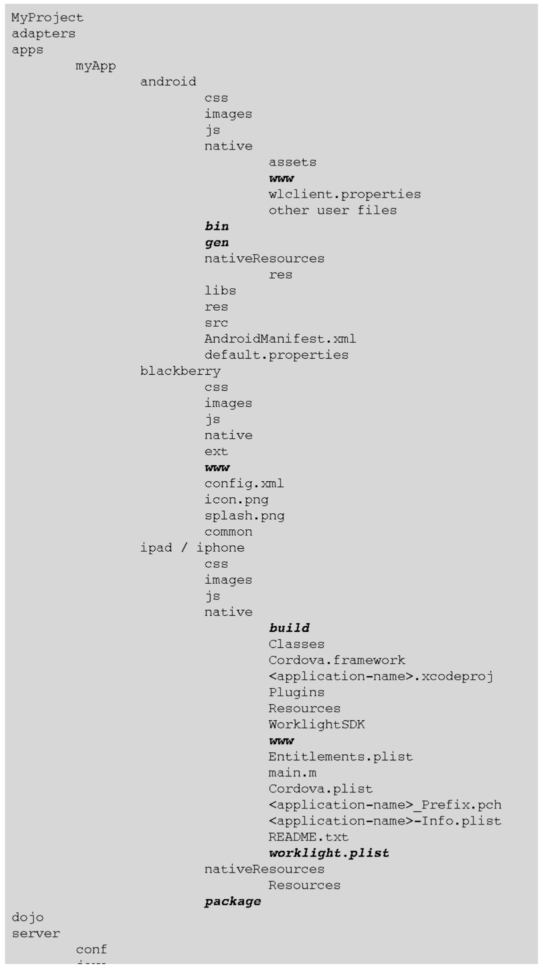
LIFECYCLE

Library Systems

- Worklight can work with most version control systems that integrate with Eclipse
- Common choices:
  - **R**ational **T**eam **C**oncert (packaged w/ WL as IBM Mobile Development Lifecycle Solution)
  - Git
  - **S**ub**v**ersio**n**

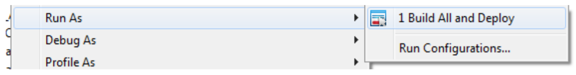
Library Systems 2

- There are files that must be excluded as they are part of WL generated resources, see here:
  - http://pic.dhe.ibm.com/infocenter/wrklight/v6r0m0/index.jsp?topic=%2Fcom.ibm.worklight.help.doc%2Fdevref%2Fr_integrating_with_source_contro.html



```
MyProject
adapters
apps
        myApp
                android
                        css
                        images
                        js
                        native
                                assets
                                www
                                wlclient.properties
                                other user files
                        bin
                        gen
                        nativeResources
                                res
                        libs
                        res
                        src
                        AndroidManifest.xml
                        default.properties
                blackberry
                        css
                        images
                        js
                        native
                        ext
                        www
                        config.xml
                        icon.png
                        splash.png
                        common
                ipad / iphone
                        css
                        images
                        js
                        native
                                build
                                Classes
                                Cordova.framework
                                <application-name>.xcodeproj
                                Plugins
                                Resources
                                WorklightSDK
                                www
                                Entitlements.plist
                                main.m
                                Cordova.plist
                                <application-name>_Prefix.pch
                                <application-name>-Info.plist
                                README.txt
                                worklight.plist
                        nativeResources
                                Resources
                        package
dojo
server
        conf
```

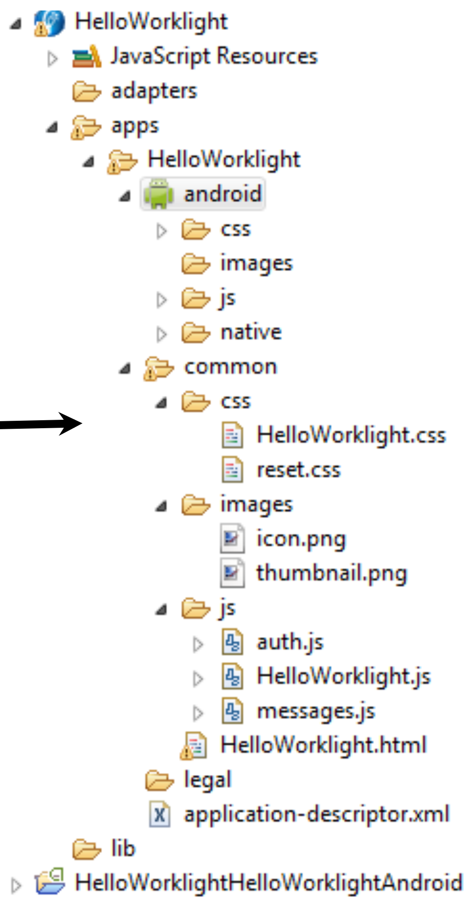
Building – Web Components



- You will want to automate your build (minification)
- **Worklight Hybrid**: Consider a pre-build approach for your web code.
  - – Faster dev time turnaround
- **Mobile Web**: Consider running a build every time, using e.g. Dojo Build: http://dojotoolkit.org/reference-guide/1.9/build/
- Running jslint / jshint to catch JS errors

Building

- Then build WL project itself
- WL provides the `<app-builder>` and `<adapter-builder>` ANT tasks
  - Only builds the Server portion of the projects - the **.war** customisation file, the **.wlapp** file, and the **.adapter** files.
  - You will need to build the **.apk** and **.ipa** files using platform-native process.



```
HelloWorklight
  JavaScript Resources
  adapters
  apps
    HelloWorklight
      android
        css
        images
        js
        native
      common
        css
          HelloWorklight.css
          reset.css
        images
          icon.png
          thumbnail.png
        js
          auth.js
          HelloWorklight.js
          messages.js
        HelloWorklight.html
      legal
      application-descriptor.xml
  lib
HelloWorklightHelloWorklightAndroid
```

Building

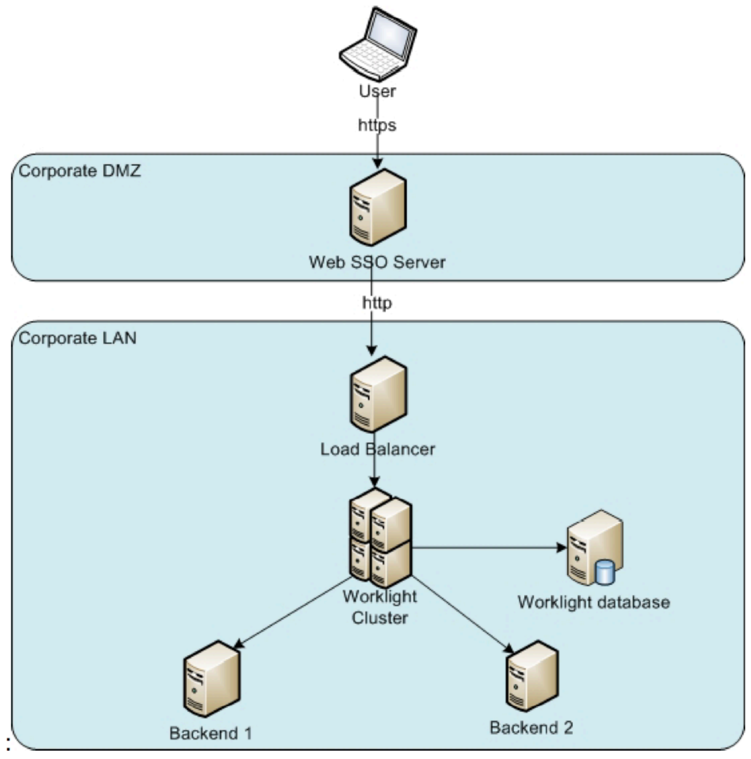
- During build, externalise certain things:
  - `worklightServerRootUrl` in `application-descriptor.xml`
  - `server/conf/worklight.properties`
  - `maxConcurrentConnectionsPerNode` for adapters
  - `domain`, `port` for the backend service in *`adapter`*`.xml`

Deploying

- Deploy the **.war** using relevant application server method
  - Whenever **server/conf/*** changes
- Deploy the **.wlapp** and **.adapter** server-side portions of the application using `<app-deployer>` and `<adapter-deployer>` ANT tasks.

Deployment Topology

- Options include:
  - *WebSphere Application Server ND* - familiar
  - *WAS Liberty Profile* – simpler, newer
- Consider **HTTPS**, **load spraying**



Deploying to Phones

- You still need to get the native application (.ipa, .apk, etc.) onto your user' s phones.
  - **Dev Time/Small/Adhoc Projects**: Manual install
  - **Testing lifecycle**: AppCenter - comes with WL server editions
    - Install via AppCenter Web or AppCenter App
  - **B2C**: public App Stores (Apple App Store, Google Play Store)
  - **B2E**: IBM Endpoint Manager or similar

Testing

- Typically you'll want to test:
  - **Manual UI** on physical phones
    - Coverage across devices
  - **Automated UI** - mocking framework and automated test tool
    - V6.0 - Mobile Test Workbench for Worklight
  - (Worklight) **Adapters** / (Mobile Web) **REST Services** - load / performance / functional tests - just HTTP

Questions?

Andrew Ferrier

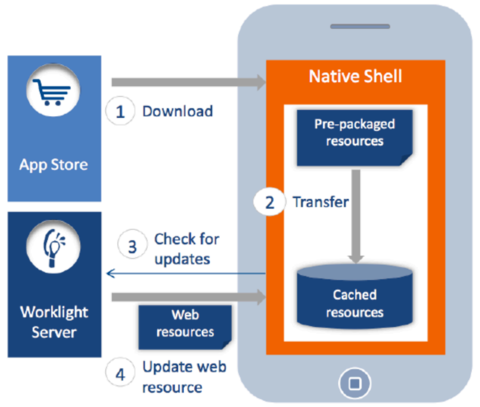
andrew.ferrier@uk.ibm.com

http://dojotipsntricks.com

UPDATES

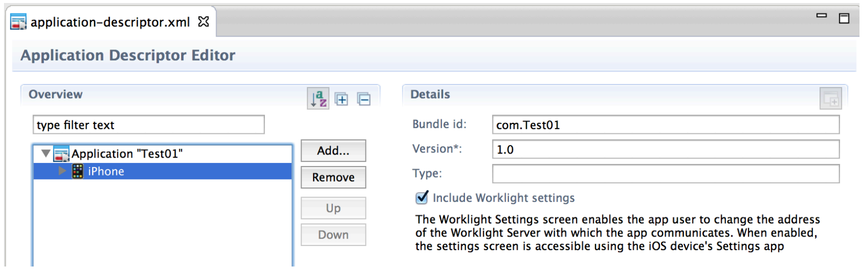
Two Ways to Update - Method 1

- Update your web code only
- **Don't** change the version number of the application
- Redeploy **.wlapp** only
- Implicitly encourages a "Direct Update" next time client connects.



Two Ways to Update - Method 2

- Method 2:
  - Update web code and custom native code
  - **Do** update the application version number
  - Re-release via binary method (App Store, etc.)



Bundle id: com.Test01
Version*: 1.0
Type:
☑ Include Worklight settings

The Worklight Settings screen enables the app user to change the address of the Worklight Server with which the app communicates. When enabled, the settings screen is accessible using the iOS device's Settings app

Updating Worklight Itself

- Upgrade all studio instances and WL environments
- All apps at all existing application versions need to be re-built (**.war**/**.wlapp**/**.adapter**)
- Re-release an app using method 2
  - Gets new Device Runtime onto end-users' phones
- But end-users can continue using old app; wire protocol is backward-compatible

OTHER TIPS & BEST PRACTICES

Client-side Worklight

- Hybrid App: Don't optimize for size of the client like you would do for Mobile Web
- Nevertheless, there is still a browser control underneath
- Use `WL.Logger.{debug,error}` API, logging in development environment is customizable, & log the username on errors

Client-side Worklight

- Understand handling errors on client-side, in particular adapter invocations:
  - http://www.ibm.com/developerworks/websphere/techjournal/1212_paris/1212_paris.html?ca=drs-
- Use `connectOnStartup: false`, with `WL.Client.Connect()` after startup - gives more startup control
  - Must be done for use of direct updates, push notifications, authentication, adapter use

Server-side Worklight JavaScript

- Discourage use of more than one adapter
  - Cannot share JavaScript code (can share Java though)
- Again, understand how to handle errors from adapter invocations (same article).
- Again, use `WL.Logger` API - has various levels of logging, can be configured on server. Log the username on errors.
  - *Note*: Log level control is currently limited with WAS