Agile Mobile Apps for the Enterprise with IBM Worklight

Agile Mobile Apps for the Enterprise with IBM Worklight

Agile Mobile Apps for the Enterprise with IBM Worklight

Andrew Ferrier
andrew.ferrier@uk.ibm.com

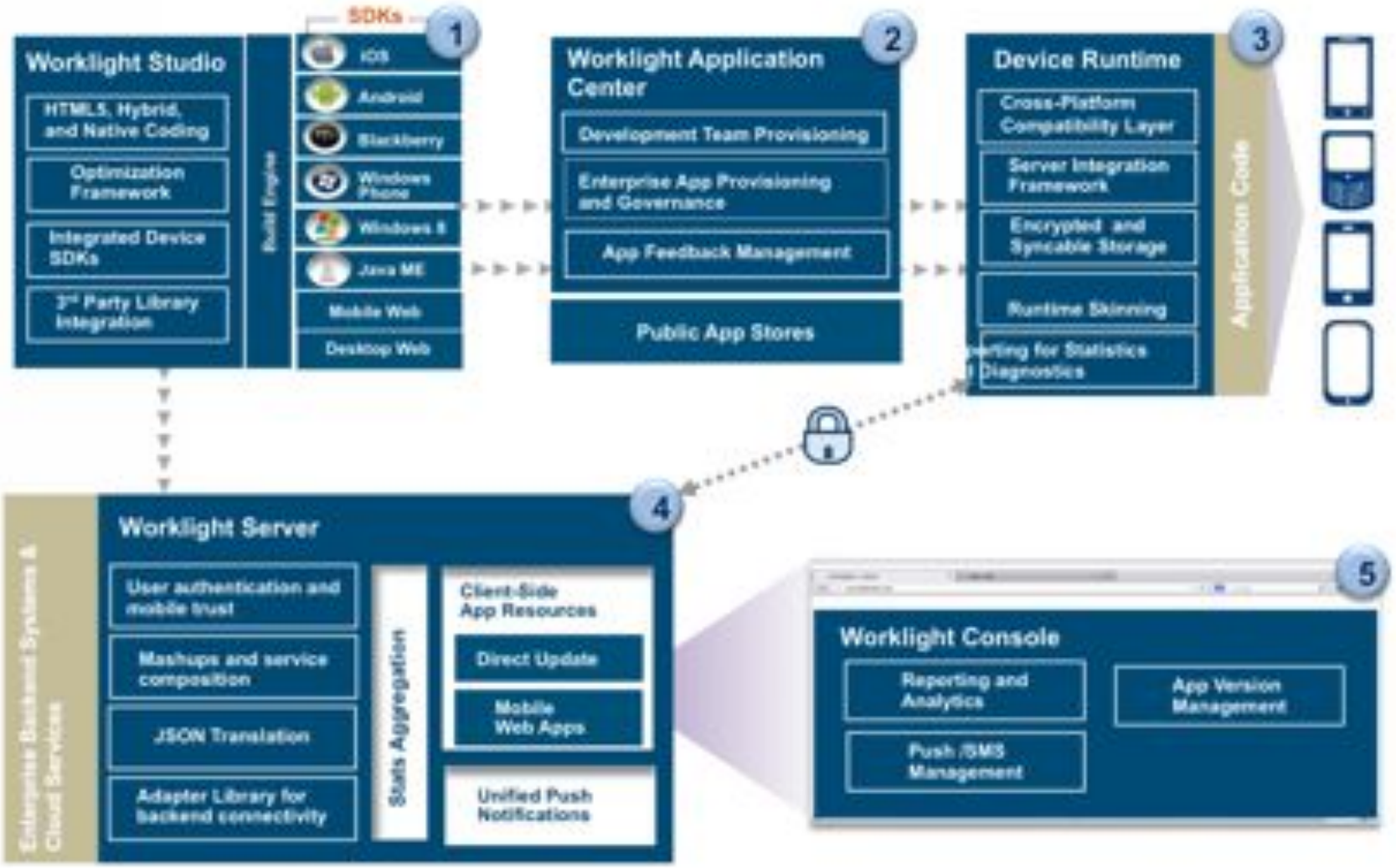
Donal Spring
donalspr@uk.ibm.com

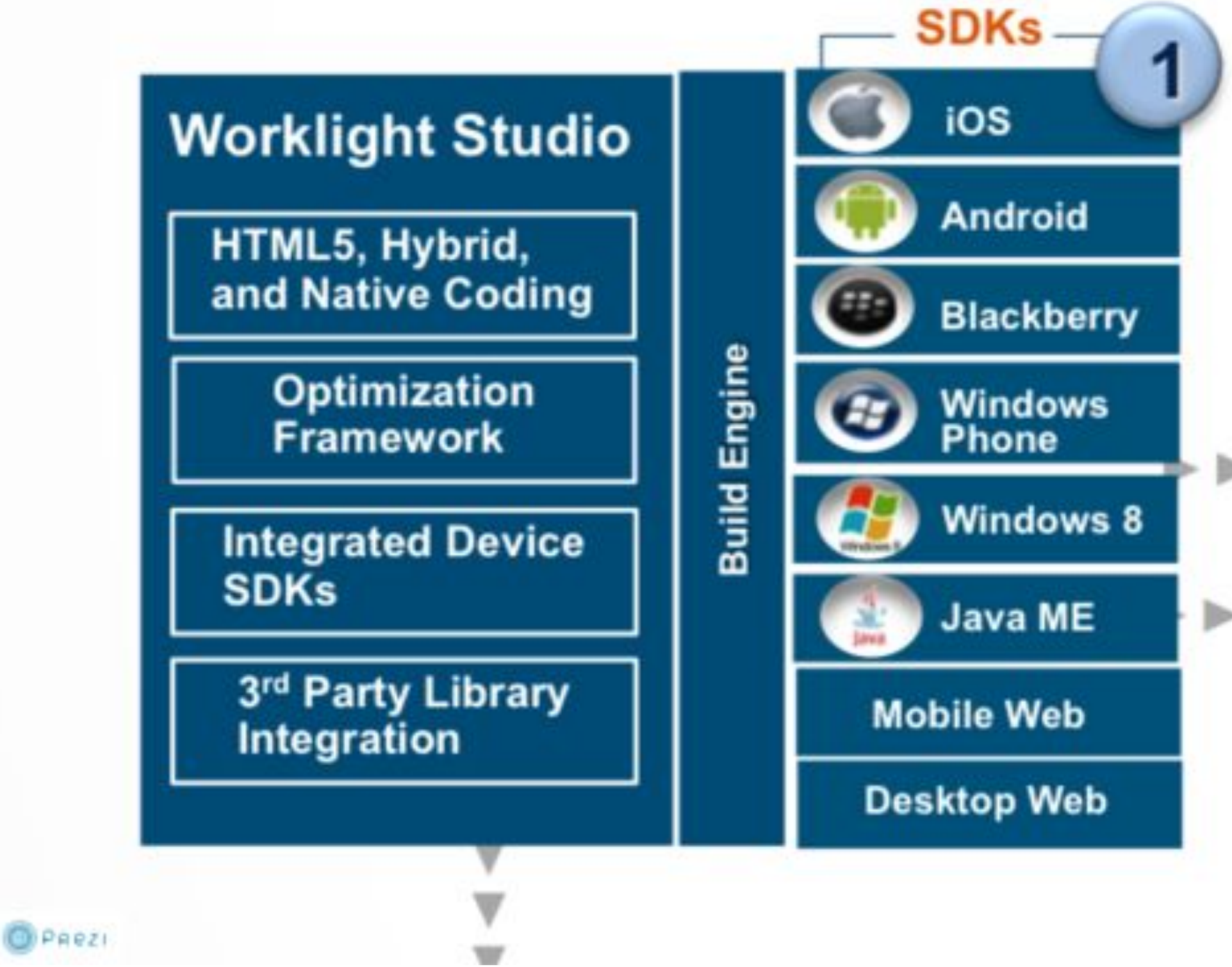
UK WebSphere User Group, 25th March 2014

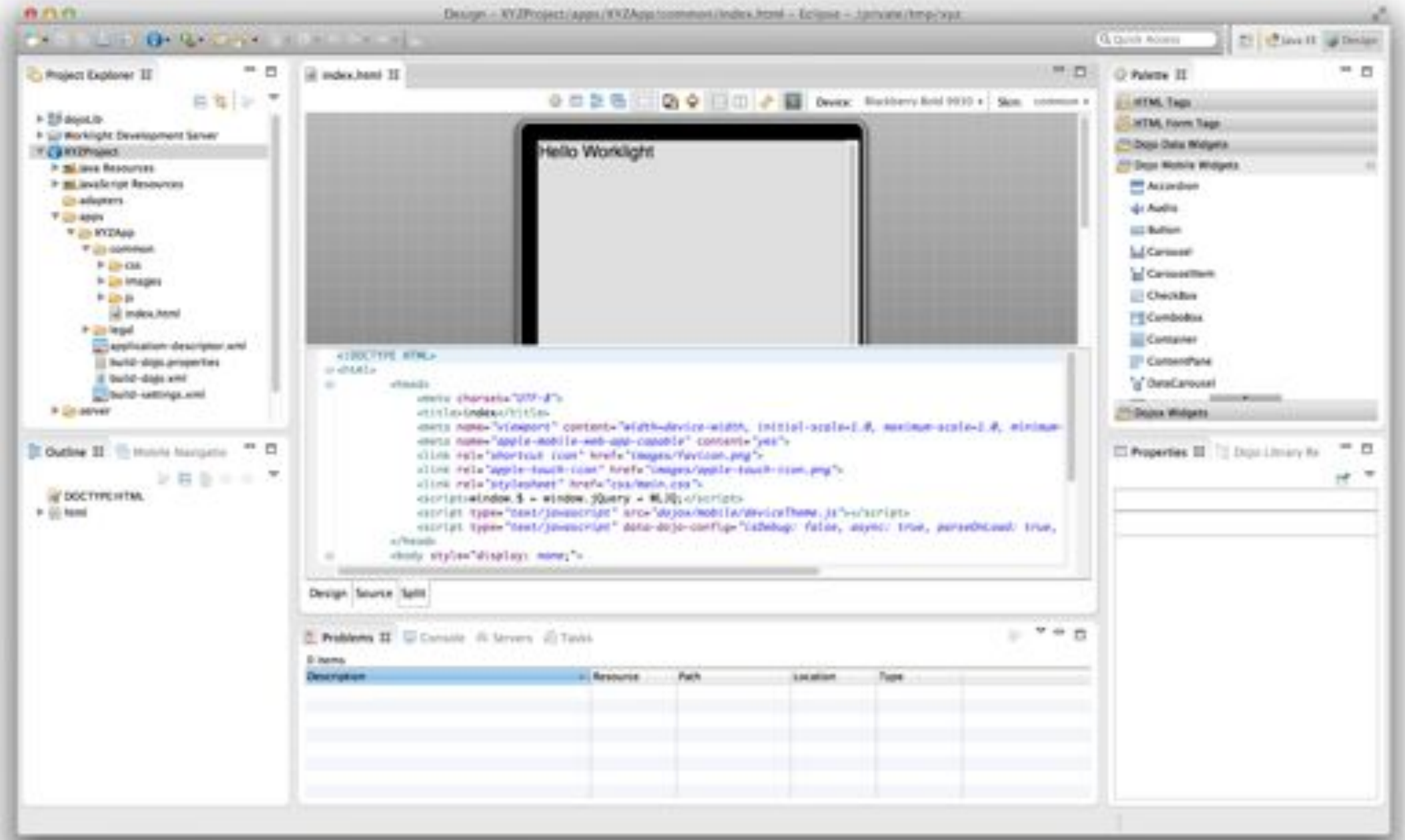
Agenda

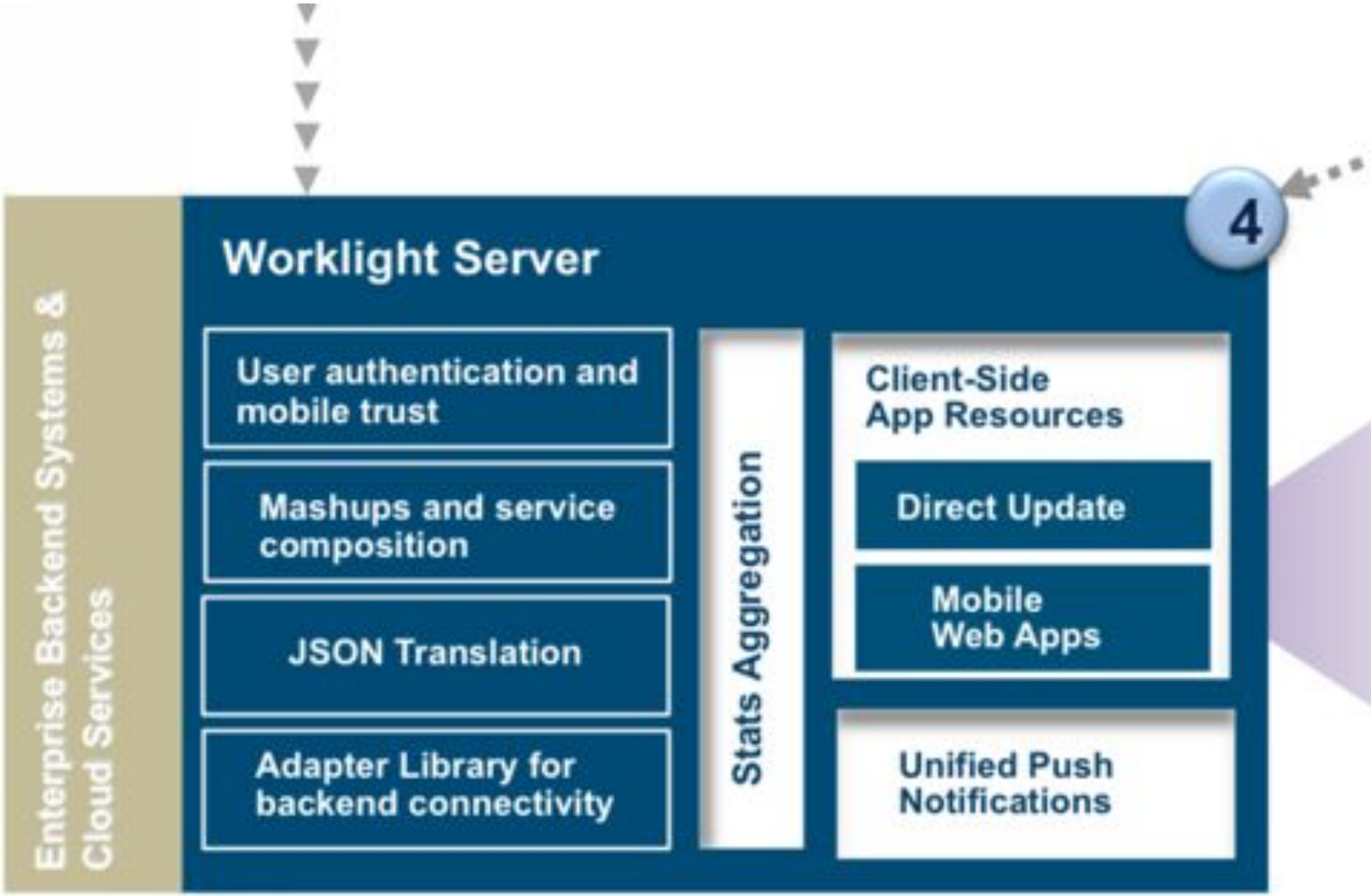
- What is IBM Worklight?: *A Review*
- Continuous Integration
- Automated Adapter Testing
- Preparing for Production (versioning adapters, managing authentication, updating apps...)

What is IBM Worklight - The Bits

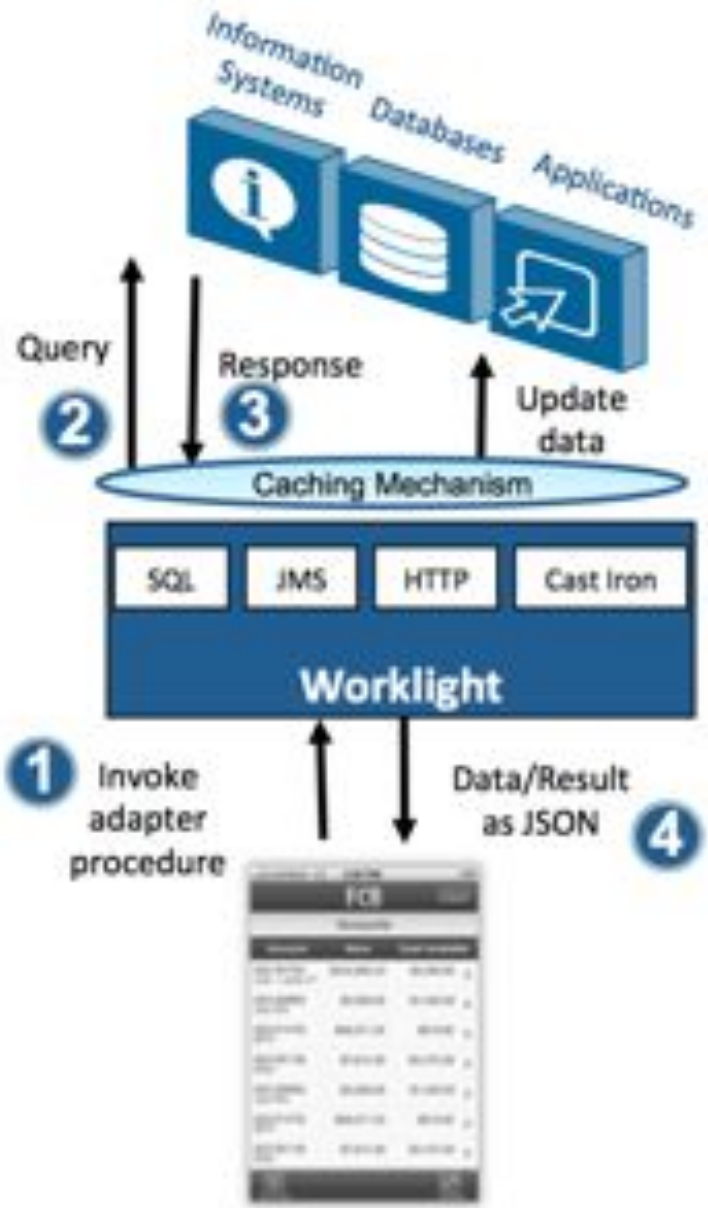



SDKs
1
Worklight Studio
HTML5, Hybrid, and Native Coding
Optimization Framework
Integrated Device SDKs
3rd Party Library Integration
Build Engine
iOS
Android
Blackberry
Windows Phone
Windows 8
Java ME
Mobile Web
Desktop Web




Enterprise Backend Systems & Cloud Services
Worklight Server
4
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

What are Worklight Adapters?



- Standardised way to "mobilise" services
- Essentially a lightweight integration layer

- Much of the real-world use is the HTTP Adapter to connect to existing RESTful Services, i.e. mapping to **Create** (POST), **Read** (GET), **Update** (PUT) and **Delete** (DELETE) verbs.
- There are also other adapter types (SQL, JMS, Cast Iron)

What do they look like?

Client-side (in the app)

```
WL.Client.invokeProcedure({
    adapter: 'Adapt',
    procedure: 'getStories',
    parameters: [],

}, {
    onSuccess: function(object) {
        // .. Do some stuff with the result
    },
    onFailure: function(object) {
        // .. Handle the error situation
    }
});
```

Server-side (on the WL Server)

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="Adapt"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wl="http://www.worklight.com/integration"
    xmlns:http="http://www.worklight.com/integration/http">

    <displayName>Adapt</displayName>
    <description>Adapt</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>http</protocol>
            <domain>rss.cnn.com</domain>
            <port>80</port>
        </connectionPolicy>
        <loadConstraints maxConcurrentConnectionsPerNode="2" />
    </connectivity>

    <procedure name="getStories"/>
</wl:adapter>
```

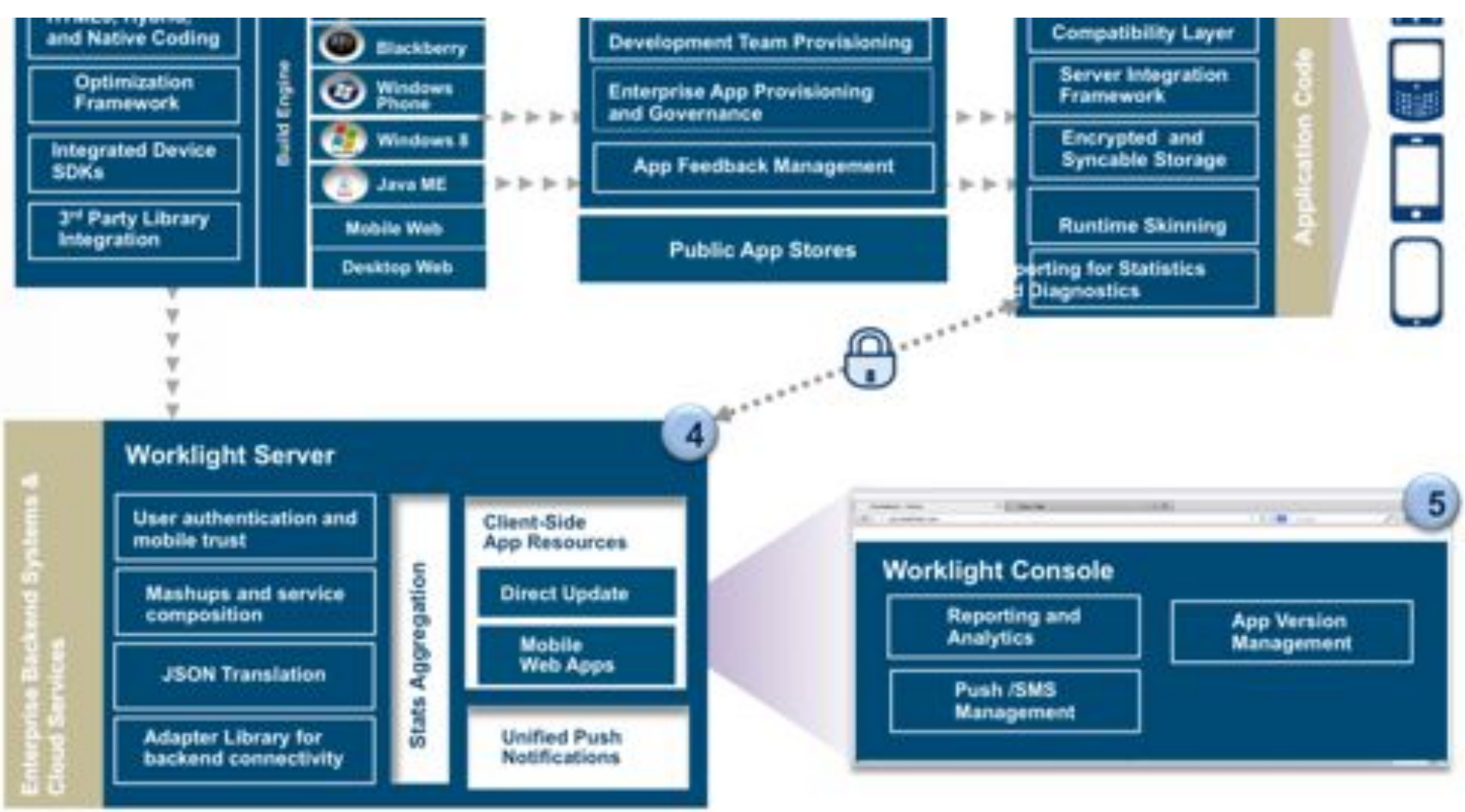
```
function getStories(param) {
    path = getPath(param);

    var input = {
        method : 'get',
        returnedContentType : 'xml',
        path : path,
        transformation : {
            type : 'xslFile',
            xslFile : 'filtered.xsl'
        }
    };

    return WL.Server.invokeHttp(input);
}
```

```
WL.Client.invokeProcedure({
    adapter: 'Adap1',
    procedure: 'getStories',
    parameters: [],

}, {
    onSuccess: function(object) {
        // .. Do some stuff with the result
    },
    onFailure: function(object) {
        // .. Handle the error situation
    }
});
```

Server side (on the WL Serv

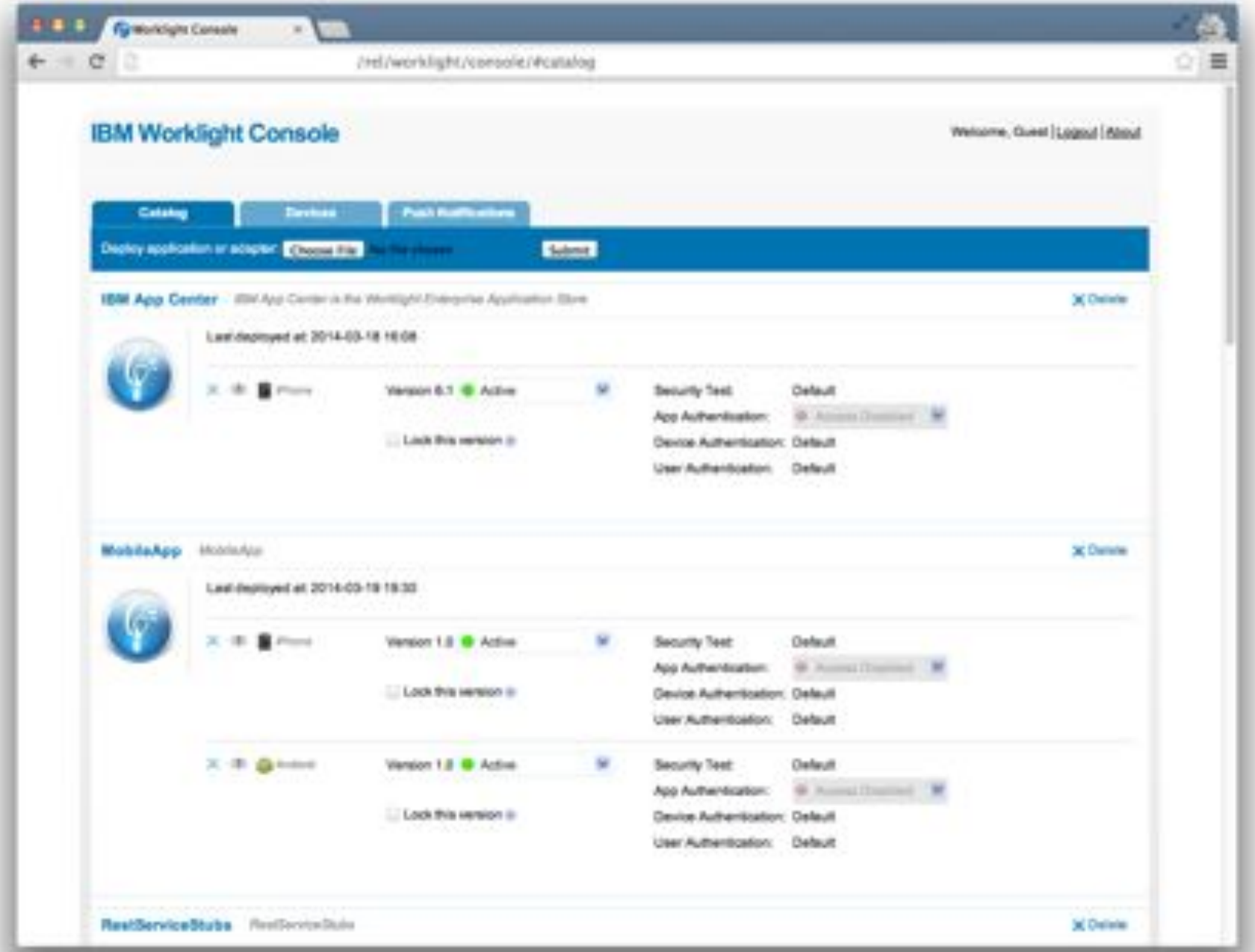
```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="Adap1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wl="http://www.worklight.com/integration"
    xmlns:http="http://www.worklight.com/integration/http">

    <displayName>Adap1</displayName>
    <description>Adap1</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>http</protocol>
            <domain>rss.cnn.com</domain>
            <port>80</port>
        </connectionPolicy>
        <loadConstraints maxConcurrentConnectionsPerNode="2" />
    </connectivity>

    <procedure name="getStories"/>
</wl:adapter>
```

```
function getStories(param) {
    path = getPath(param);

    var input = {
        method : 'get',
        returnedContentType : 'xml',
        path : path,
        transformation : {
            type : 'xslFile',
            xslFile : 'filtered.xsl'
        }
    };

    return WL.Server.invokeHttp(input);
}
```

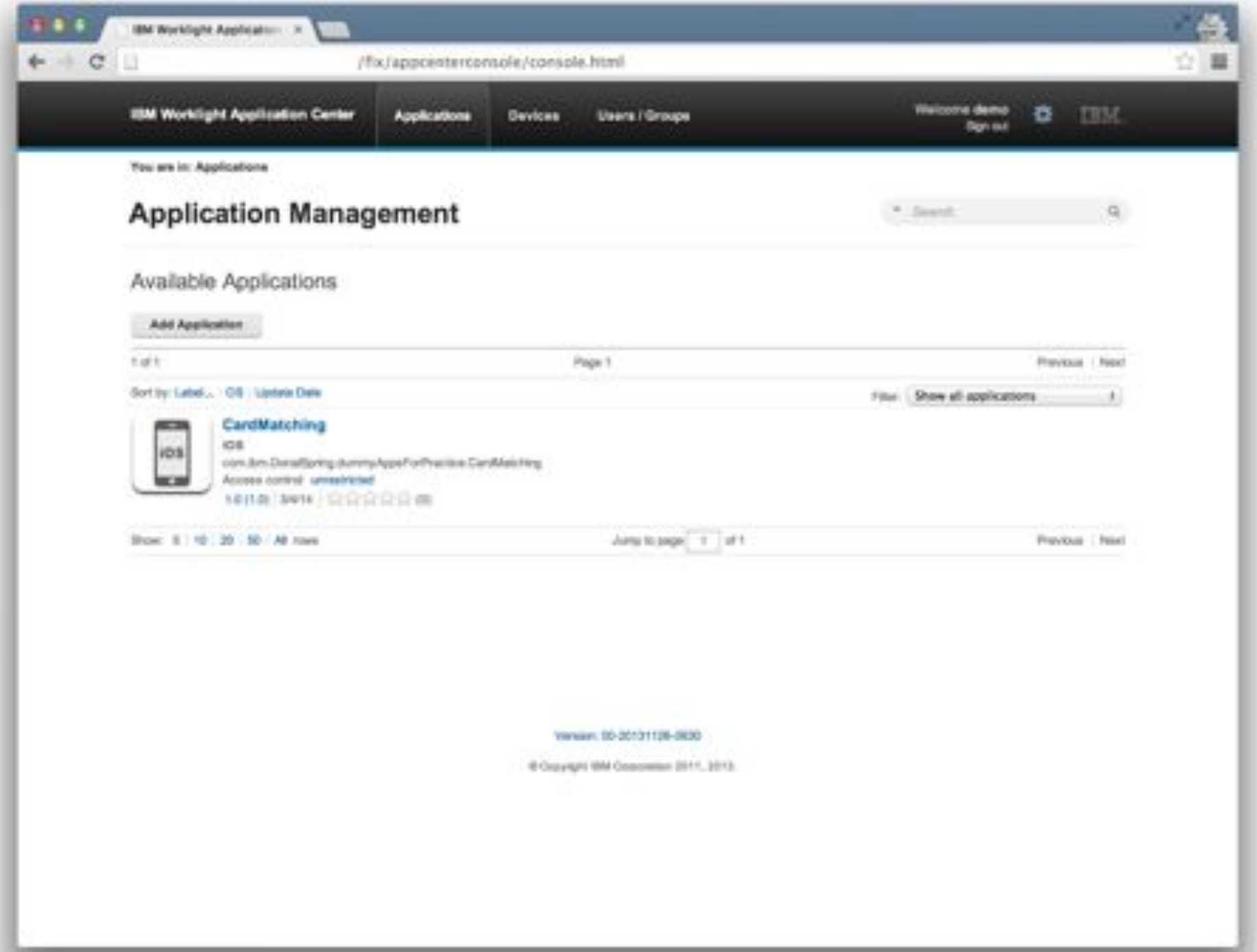


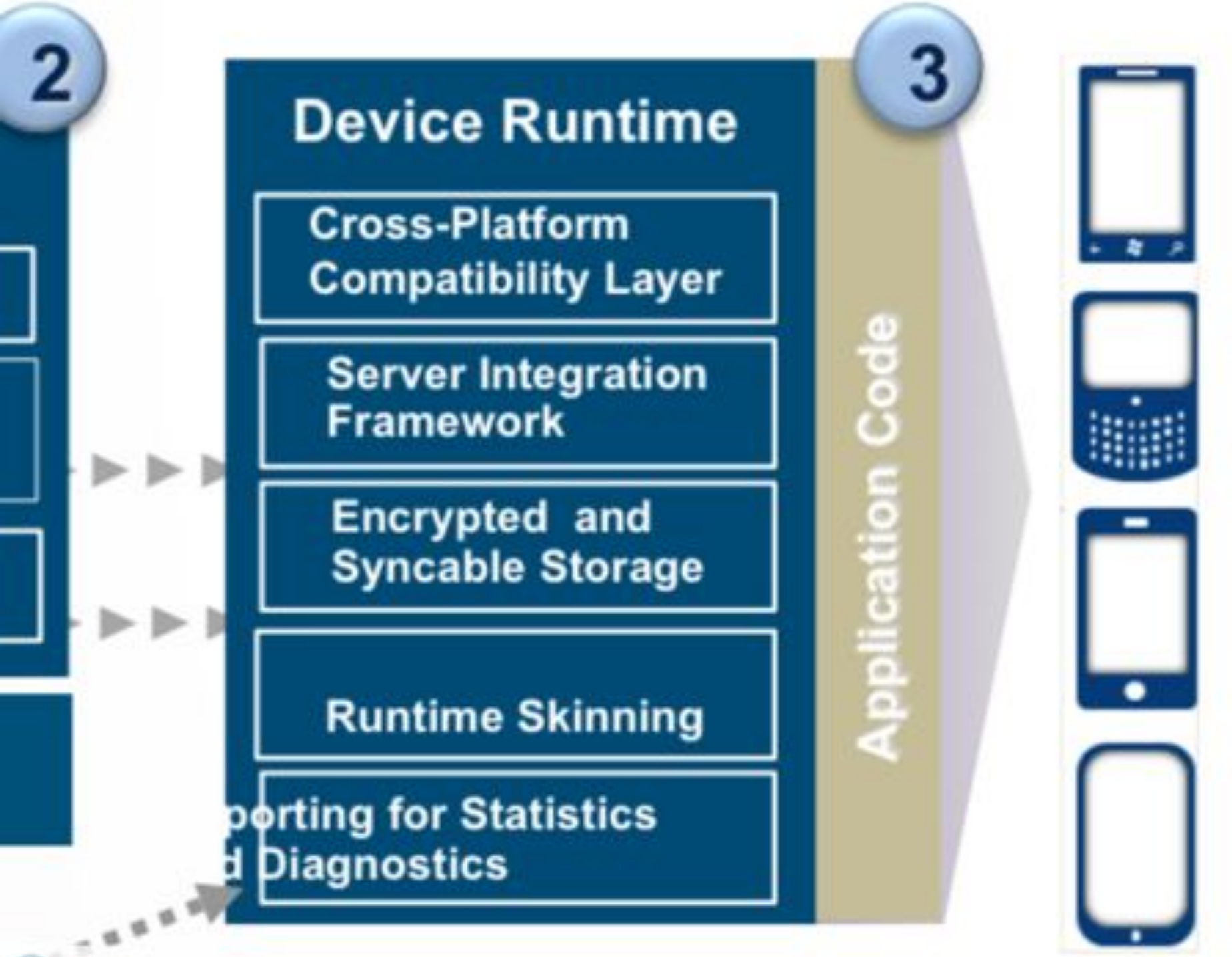
Prezi

Worklight Adapters?


Worklight Console
/rel/worklight/console/#catalog
IBM Worklight Console
Welcome, Guest | Logout | About
Catalog
Devices
Push Notifications
Deploy application or adapter:
Choose File
No file chosen
Submit
IBM App Center
Delete
Last deployed at: 2014-03-18 16:08
Version 6.1
Active
Lock this version
Security Test:
Default
App Authentication:
Device Authentication:
Default
User Authentication:
Default
MobileApp
Delete
Last deployed at: 2014-03-19 19:33
Version 1.0
Active
Lock this version
Security Test:
Default
App Authentication:
Device Authentication:
Default
User Authentication:
Default
Version 1.0
Active
Lock this version
Security Test:
Default
App Authentication:
Device Authentication:
Default
User Authentication:
Default
RestServiceStubs
Delete


2
Worklight Application Center
Development Team Provisioning
Enterprise App Provisioning and Governance
App Feedback Management
Public App Stores


/fix/appcenterconsole/console.html
IBM Worklight Application Center
Applications
Devices
Users / Groups
Application Management
Available Applications
Add Application
CardMatching
iOS

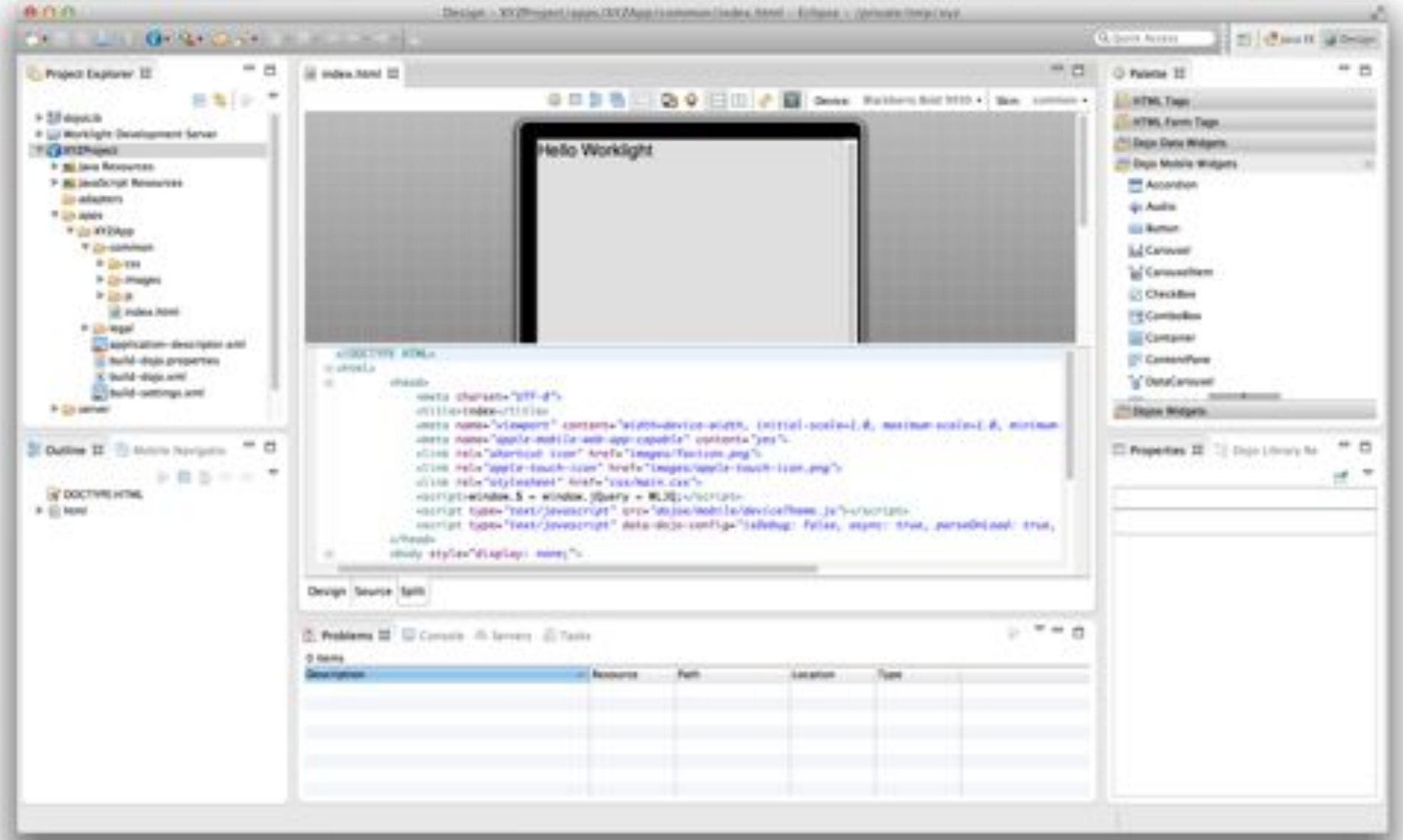

2
Device Runtime
Cross-Platform Compatibility Layer
Server Integration Framework
Encrypted and Syncable Storage
Runtime Skinning
porting for Statistics
d Diagnostics
3
Application Code


File System
(on mobile device)
Native Container
HTML, CSS, JavaScript
Mobile Operating System


Native Container
Mobile Operating System
Worklight Server
(Adapters)
Back-end Systems
(HTTP, REST, SOAP,
JDBC, etc...)

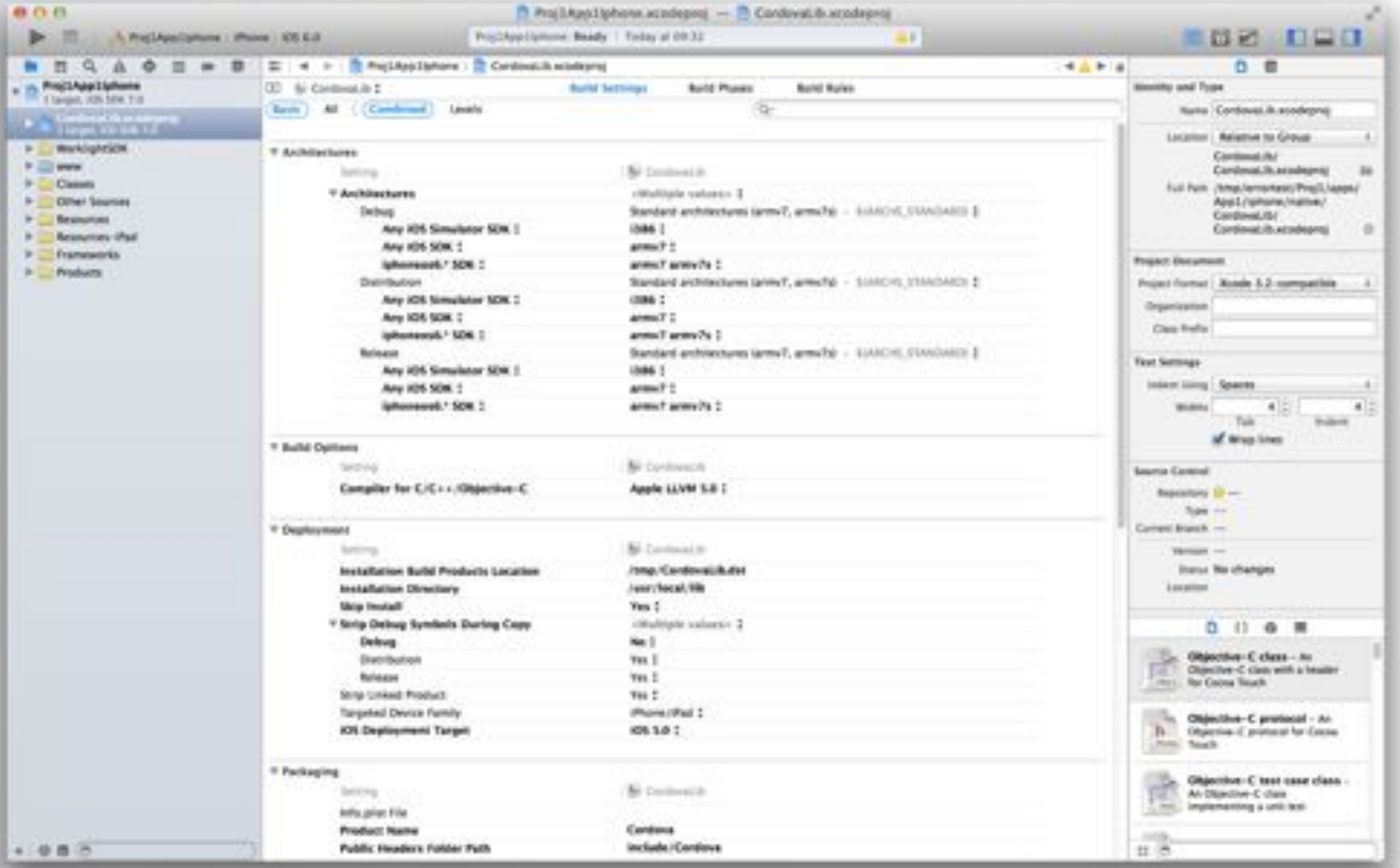
So how would we "normally" develop a Worklight app?


Develop App and
Adapters
... using the IDE


Hello Worklight

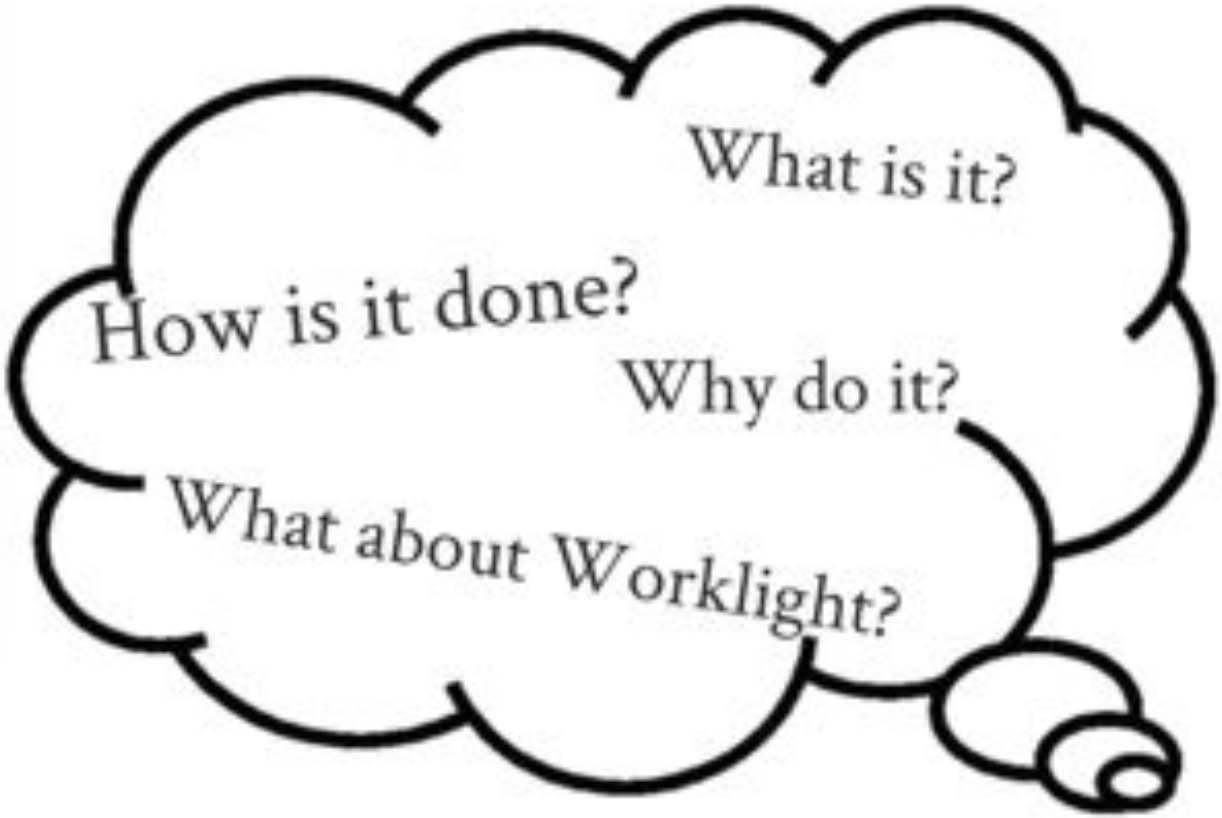
Use Platform-native Tools to Build

Tools to Build



Deploy Artifacts
Manually for
Development and
Testing

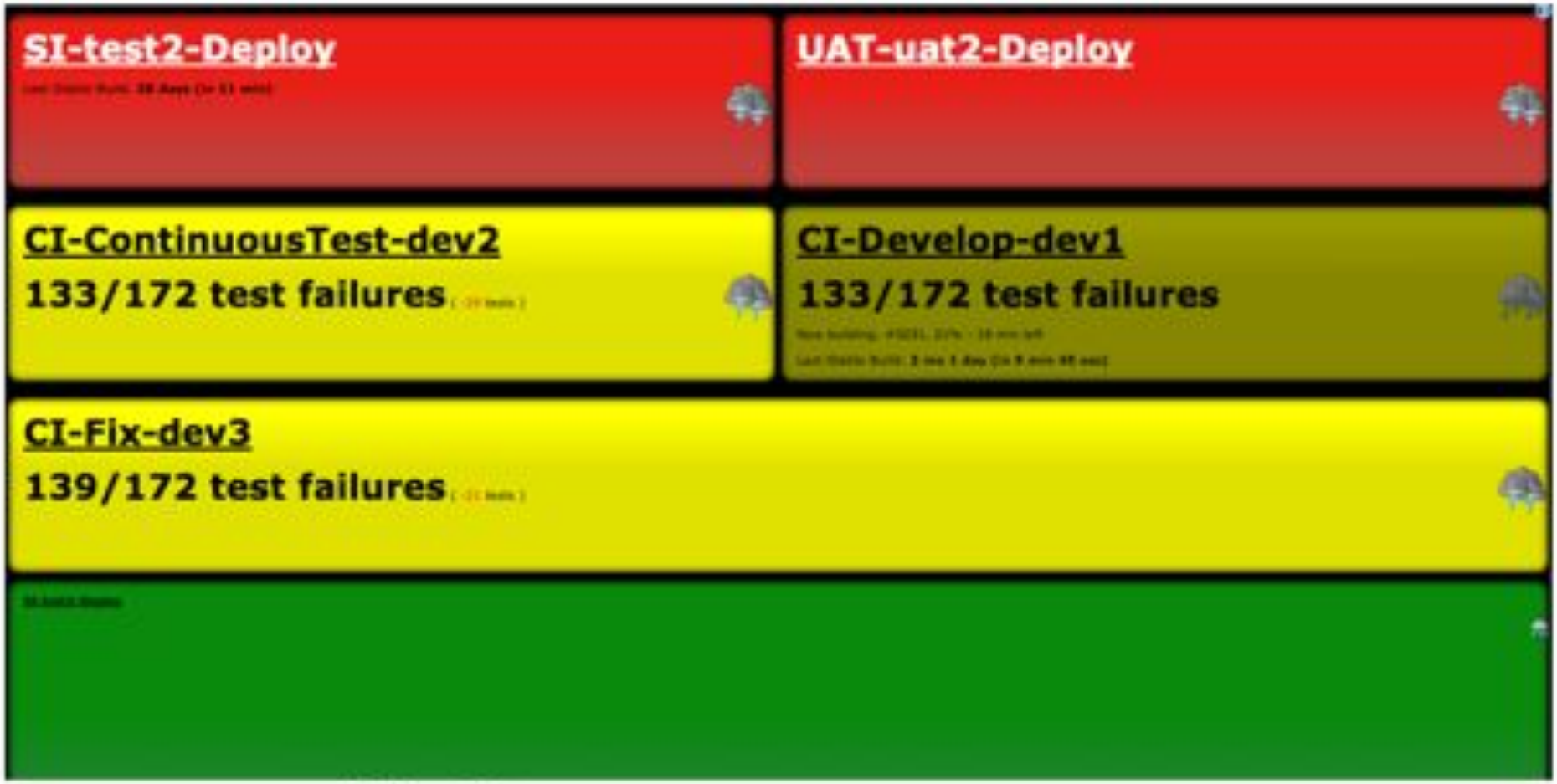
Continuous Integration



Build in the morning have your fix on devices by the end of the day

Jenkins

Why would we do it? This is why:



Client transparency

Time to market

Encourages developers

Mitigates integration issues

Ideal Steps to Production

CI Environment

- Development:
  - [illegible] frequently the latest [illegible] of code checked into SCM
  - possibly unstable, provides immediate feedback

(Future Proofing)

- Bug Fixing - Once version 1.0 of the app is in app store and v1.1 is being created. Fixes can be made to the app in parallel with future development

SI Environment

Systems Integration:

- More stable code base
- Promoted at end of Sprints
- End to end process tests
- QA and UI testing can be expanded
- Integration tests
- Another App Center

UAT Environment

User Acceptance Testing

- The final check before code is pushed to production
- Business sign off
- Final QA
- "Production like" environment (clustered, load balanced etc)

Production ready app in App Stores

CI Environment

- Development:
  - built very frequently the latest level of code checked into SCM
  - possibly unstable, provides immediate feedback

(Future Proofing)

- Bug Fixing - Once version 1.0 of the app is in app store and v1.1 is being created. Fixes can be made to the app in parallel with future development

CI provides immediate feedback to developers

app in parallel with future developm

CI provides immediate feedback to developers

Jenkins

Jenkins › CI-Develop-dev1 › #3171

- Back to Project
- **Status**
- Changes
- Console Output
- Edit Build Information
- Delete Build
- Polling Log
- Build Graph
- Previous Build

Build #3171 (Mar 21, 20

Build Artifacts

- Basket.adapter
- Category.adapter
- CommonUtilities.adapter
- Content.adapter
- Customer.adapter
- [redacted]war
- Location.adapter
- Product.adapter
- RestServiceStubs-mobilewebapp-1.0.wlap
- Reviews.adapter
- Search.adapter
- SecurityTestApp-mobilewebapp-1.0.wlapp
- Stock.adapter

No changes.

Started by GitHub push by MarcoSero Url Trigger
app/commit/0258d63ace7dab7b46d7c5b43e5c23

Jenkins

Jenkins › SI-test1-Deploy › #133

- Back to Project
- **Status**
- Changes
- Console Output [raw]
- Edit Build Information
- Delete Build
- Parameters
- Build Graph
- Previous Build

Build #133 (Mar 19, 2

No changes.

Started by anonymous user

Continuous Integration server

Automated ways of building each component

What tools do we need?

Mature source-code management is vital e.g. git.

A few bits of glue in the form of scripts etc.

Automated Testing?

+

Liberty profile and Jenkins is a free, open source and highly customisable solution

Continuous Integration server

Automated ways of building each component

What tools do we need?

Mature source-code management is vital e.g. git.

A few bits of glue in the form of scripts etc.

Automated Testing?

The Components of Worklight

- Worklight server-side components
  - .war - The Worklight server
  - .wlapp - The Worklight application
  - .adapter - The adapter

Worklight supplies Ant Tasks to build & deploy its components

Continuous Integration server

Automated ways of building each component

What tools do we need?

Mature source-code management is vital e.g. git.

A few bits of glue in the form of scripts etc.

Automated Testing?

Powerful Source-code Management

- GitHub Pull requests or similar features enable easy code peer review prior to integration

- GitHub Webhooks can trigger builds on code commit allowing CI to "*blame*" code mergers that break the build

Powerful SCM can boost development time and simplify integration

All Requests 1

Yours

Find a user...

andrew123 1

Open | Closed

Sort: Newest

New pull request

Cleanup whitespace. #171

I would like to clean up the whitespace in worklight.properties, please...

by andrew123 3 minutes ago feature/cleanup-whitespace

Keyboard shortcuts available

Cleanup whitespace. #171

Edit

Open andrew123 wants to merge 1 commit into develop from feature/cleanup-wht...

Conversation 0 Commits 1 Files Changed 1

Showing 1 changed file with 7 additions and 7 deletions.

Show Diff Stats

Homebase/server/conf/worklight.properties Open Edit View

```
@@ -12,7 +12,7 @@
# In worklight.properties you define:
#
#   - my.adapter.db.jndi.name=jdbc/MyAdapterDS
-#
+
# In adapter.xml:
#
#  <wl:adapter>
@@ -23,7 +23,7 @@
#       <connectionPolicy xsi:type="wl:JDBCConnectionPolicy">
#           <dataSourceJNDIName>${my.adapter.db.jndi.name}</dataSourceJNDIName>
#       </connectionPolicy>
-#
+
# Both properties will be exposed as JNDI entries for the project name&#8209;application.WAR file.
```

Powerful Source-code Management

- GitHub Pull requests or similar features enable easy code peer review prior to integration

- GitHub Webhooks can trigger builds on code commit allowing CI to "*blame*" code mergers that break the build

Powerful SCM can boost development time and simplify integration

Continuous Integration server

Automated ways of building each component

What tools do we need?

Mature source-code management is vital e.g. git.

A few bits of glue in the form of scripts etc.

Automated Testing?

The glue

- Scripts and Ant tasks to
  - Source Code Checkout
  - Build Worklight artefacts
  - Deploy .wlapp & .adapters
  - Build native applications (.ipa, etc)
  - Execute tests
  - Deploy Apps to app center
- CI server to drive & manage these

Ideal Steps to Production

CI Environment

- Development
  - built very frequently the latest level of code checked into SCM
  - possibly unstable, provides immediate feedback

(Future Proofing)

- Bug Fixing - Once version 1.0 of the app is in app store and v1.1 is being created. Fixes can be made to the app in parallel with future development

SI Environment

Systems Integration:

- More stable code base
- Promoted at end of Sprints
- End to end process tests
- QA and UI testing can be expanded
- Integration tests
- Another App Center

UAT Environment

User Acceptance Testing

- The final check before code is pushed to production
- Business sign off
- Final QA
- "Production like" environment (clustered, load balanced etc)

Production ready app in App Stores

SI Environment

Systems Integration:

- More stable code base
- Promoted at end of Sprints
- End to end process tests
- QA and UI testing can be expanded
- Integration tests
- Another App Center

UAT Environment

User Acceptance Testing

- The final check before code is pushed to production
- Business sign off
- Final QA
- "Production like" environment (clustered, load balanced etc)

Ideal Steps to Production

CI Environment

- Development:
  - built very frequently the latest level of code checked into SCM
  - possibly unstable, provides immediate feedback

(Future Proofing):

- Bug Fixing - Once version 1.0 of the app is in app store and v1.1 is being created. Fixes can be made to the app in parallel with future development

SI Environment

Systems Integration:

- More stable code base
- Promoted at end of Sprints
- End to end process tests
- QA and UI testing can be expanded
- Integration tests
- Another App Center

UAT Environment

User Acceptance Testing

- The final check before code is pushed to production
- Business sign off
- Final QA
- "Production like" environment (clustered, load balanced etc)

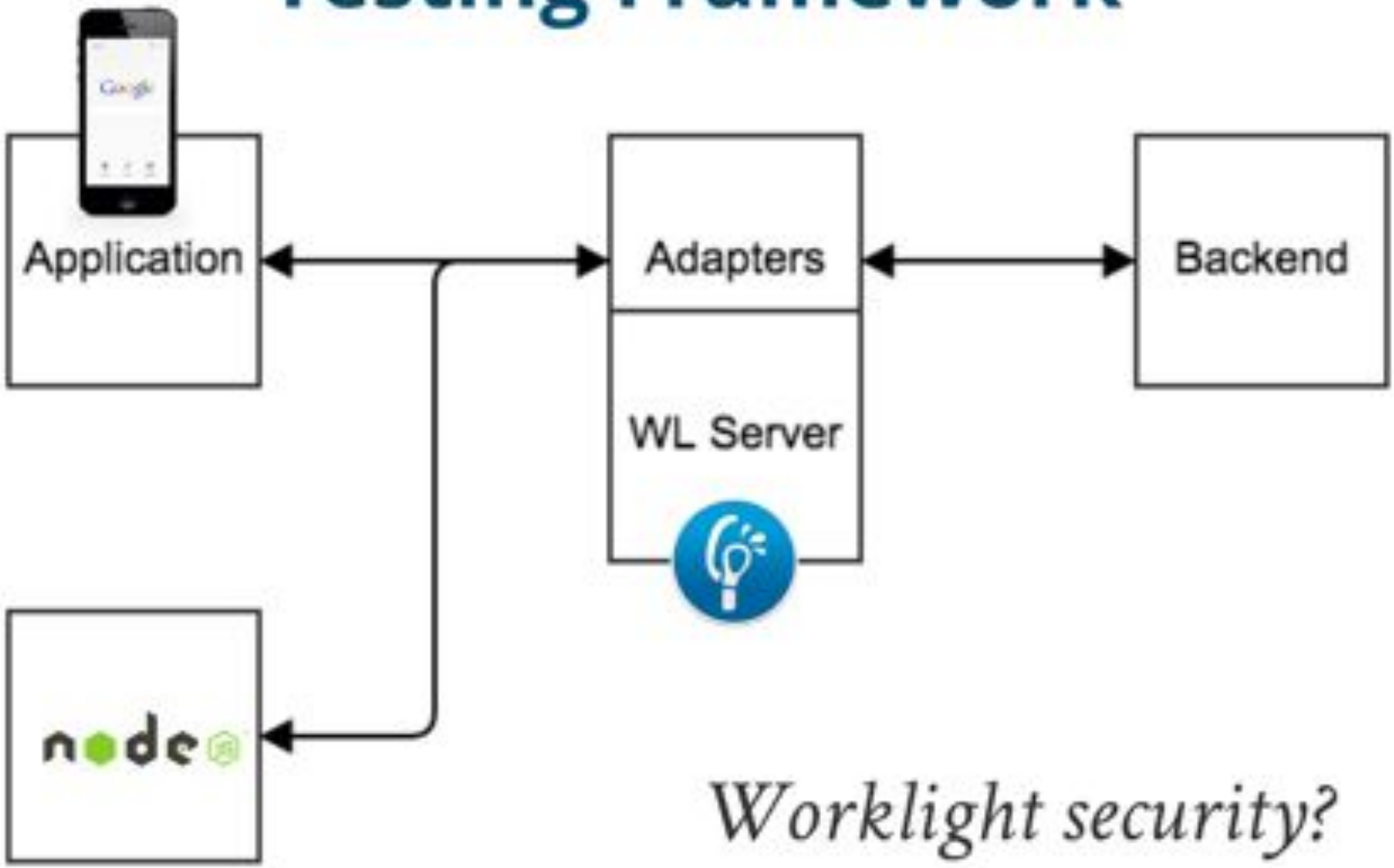
Production ready app in App Stores

Automated Adapter Testing

- Testing server-side components with well-defined inputs and outputs.
- Provides feedback on code quality and data manipulation
- Integration testing of adapters supported

- Integrated into CI - this gives a powerful metric
- We have successfully combined open-source components with the Worklight API to construct such a framework.

Testing Framework



Worklight security?

Automated Adapter Testing

- Testing server-side components with well-defined inputs and outputs.
- Provides feedback on code quality and data manipulation
- Integration testing of adapters supported

- Integrated into CI - this gives a powerful metric
- We have successfully combined open-source components with the Worklight API to construct such a framework.

What tools do we need?

simple, flexible, fun

- Customised Worklight node script to invoke the adapters
- Stubbing solution - Use another Worklight Mobile Web Project
  - Worklight can serve up text in response to GET requests

```
1  var ip = require('wlAdapter');
2  var expect = require('chai').expect;
3
4  var param = "[]";
5
6  describe("Category Adapter Test Suite", function(){
7      describe ("#getTopLevelCategories() ", function(){
8          it("should have data field containing something", function(done){
9              ip.invokeProcedure("Category", "getTopLevelCategories", param, function(response){
10                 expect(response.data).not.to.be.null;
11             },done);  // end callback
12         }); // end it
13     }); // end describe
14 }); // end describe
```

```
~ UnitTest git:(devwlop) ✗ mocha -R spec tests/getTopLevelCategories.testJ.demo.js


  Category Adapter Test Suite
    #getTopLevelCategories()
      ✓ should have data field containing something (189ms)


  1 passing (189ms)

~ UnitTest git:(devwlop) ✗ mocha -R spec tests/getTopLevelCategories.testJ.demo.js


  Category Adapter Test Suite
    #getTopLevelCategories()
      1) should not have data field


  0 passing (123ms)
  1 failing

  1) Category Adapter Test Suite #getTopLevelCategories()  should not have data field:
     Error: expected { Object (statusCode, Search, ...) } to be null
      at reportTestFailure (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/wlAdapter/invokeWLProcedure.js:230:13)
      at /Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/wlAdapter/invokeWLProcedure.js:35:7
      at tryCatchReject (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/makePromise.js:863:14)
      at Handler.RejectedHandler.when (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/makePromise.js:916:7)
      at Handler.deferredHandler.run (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/makePromise.js:384:13)
      at Scheduler._drainQueue (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/scheduler.js:62:14)
      at Scheduler.drainQueue (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/scheduler.js:39:8)
      at process._tickCallback (node.js:415:13)


~ UnitTest git:(devwlop) ✗
```

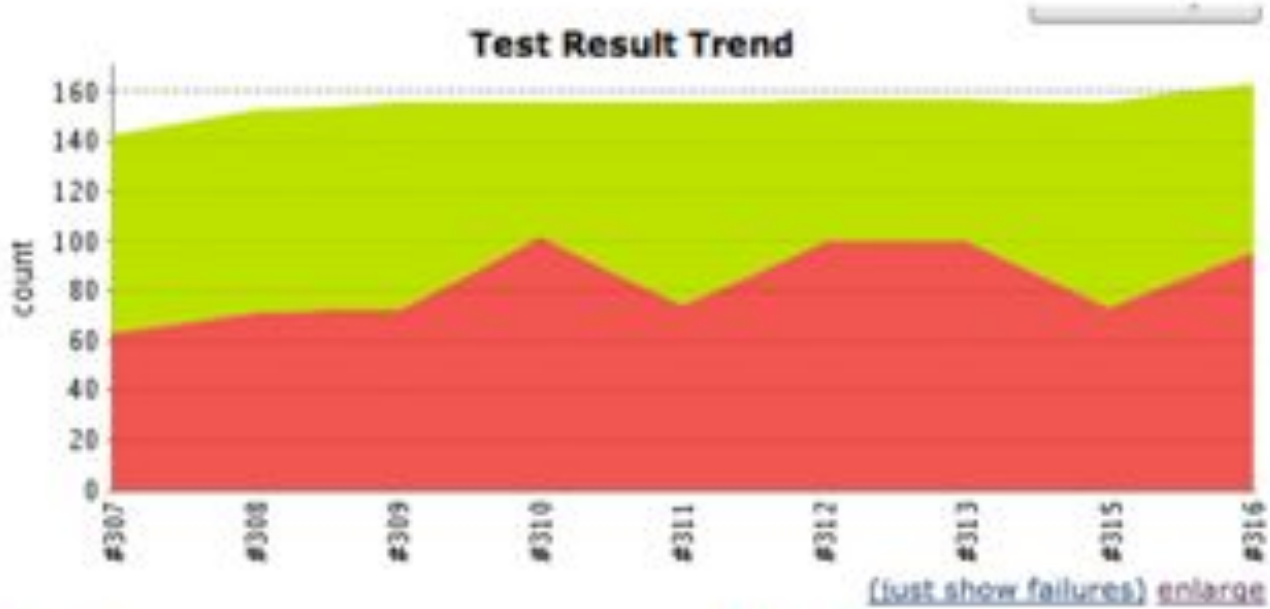
```
1  var ip = require('wlAdapter');
2  var expect = require('chai').expect;
3
4  var param = "[]";
5
6  describe("Category Adapter Test Suite", function(){
7      describe ("#getTopLevelCategories() ", function(){
8          it("should have data field containing something", function(done){
9              ip.invokeProcedure("Category", "getTopLevelCategories", param, function(response){
10                 expect(response.data).not.to.be.null;
11             },done);  // end callback
12         }); // end it
13     }); // end describe
14 }); // end describe
```

```
- UnitTest git:(develop) ✗ mocha -R spec tests/getTopLevelCategories.test.demo.js

Category Adapter Test Suite
  #getTopLevelCategories()
    should have data field containing something (185ms)
```

```
→  UnitTest git:(develop) ✗ mocha -R spec tests/getTopLevelCategories.test.demo.js


  Category Adapter Test Suite
    #getTopLevelCategories()
      ✓ should have data field containing something (105ms)


  1 passing (108ms)

→  UnitTest git:(develop) ✗ mocha -R spec tests/getTopLevelCategories.test.demo.js


  Category Adapter Test Suite
    #getTopLevelCategories()
      1) should not have data field


  0 passing (123ms)
  1 failing

  1) Category Adapter Test Suite #getTopLevelCategories()  should not have data field:
     Error: expected { Object (statusCode, Search, ...) } to be null
      at reportTestFailure (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/wlAdapter/invokeWLProcedure.js:230:13)
      at /Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/wlAdapter/invokeWLProcedure.js:35:7
      at tryCatchReject (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/makePromise.js:862:14)
      at Handler.RejectedHandler.when (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/makePromise.js:554:7)
      at Handler.DeferredHandler.run (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/makePromise.js:384:13)
      at Scheduler._drainQueue (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/scheduler.js:42:14)
      at Scheduler.drainQueue (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modules/when/lib/scheduler.js:19:9)
      at process._tickCallback (node.js:415:13)


→  UnitTest git:(develop) ✗
```

```
→  UnitTest git:(develop) ✗ mocha -R spec tests/getTopLevelCategories.test.demo.js


  Category Adapter Test Suite
    #getTopLevelCategories()
      ✓ should have data field containing something (105ms)


  1 passing (109ms)

→  UnitTest git:(develop) ✗ mocha -R spec tests/getTopLevelCategories.test.demo.js


  Category Adapter Test Suite
    #getTopLevelCategories()
      1) should not have data field


  0 passing (123ms)
  1 failing

  1) Category Adapter Test Suite #getTopLevelCategories() should not have data field:
```

```
  1 passing (109ms)

→  UnitTest git:(develop) ✗ mocha -R spec tests/getTopLevelCategories.test.demo.js


  Category Adapter Test Suite
    #getTopLevelCategories()
      1) should not have data field


  0 passing (123ms)
  1 failing

  1) Category Adapter Test Suite #getTopLevelCategories()  should not have data field:
     Error: expected { Object (statusCode, Search, ...) } to be null
      at reportTestFailure (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters
      at /Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/UnitTest/node_modu
      at tryCatchReject (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapters/Un
      at Handler.RejectedHandler.when (/Users/Marland/git/co.uk.homebase.strategic-app/Homeba
      at Handler.DeferredHandler.run (/Users/Marland/git/co.uk.homebase.strategic-app/Homebas
      at Scheduler._drainQueue (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adap
      at Scheduler.drainQueue (/Users/Marland/git/co.uk.homebase.strategic-app/Homebase/adapt
      at process._tickCallback (node.js:415:13)


→  UnitTest git:(develop) ✗
```



Jenkins can read test results

Jenkins

Jenkins › CI-Develop-dev1 › #3124 › Test Results

Back to Project
Status
Changes
Console Output [raw]
Edit Build Information
History
Polling Log
Test Result

Test Result

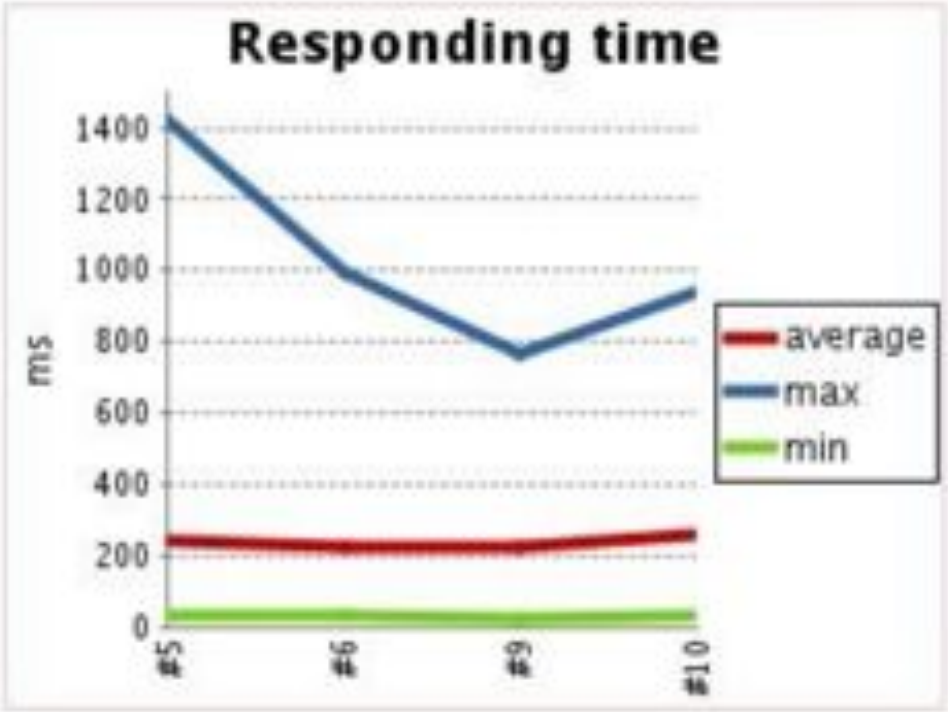
111 failures (+111)

All Failed Tests

Test Name
Content Adapter Test Suite #getContent() should work if statusCode is 200 - UAT1 version should work if statusCode is 200

Performance Testing

- Performance testing the adapter layer against stubs then against real services.
- Provides instant feedback on performance regressions and improvements



Food for thought...

- How do you handle PUT, POST, and DELETE?
- What do you do with tests that connect to both stubbed data and real services?
- How are unit tests different from integration tests?

Preparing for Production

(or, a few gotchas...)

- Worklight Applications have good management for versioning
- However, currently Worklight Adapters **do not**
- Therefore, you should do it **in code** before you reach v1, to prepare yourself for v2:

```
WL.Client.invokeProcedure({
    adapter: "Adapter1",
    procedure: "getStories",
    parameters: [1, "someOtherData"],
});
```

Understand Updates

Two ways of updating:

1. Update web code only - redeploy .wlapp only

Implictly encourages "Direct Update"

Suitable primarily for B2E scenarios

App Store

1 Download

Worklight Server

3 Check for updates

Web resources

4 Update web resource

Native Shell

Pre-packaged resources

2 Transfer

Cached resources

Understand Updates

2. • Update web code and (custom) native code
   • Re-release via consumer App Store

   • More 'heavyweight', suitable for all scenarios

Preparing for Production

(or, a few gotchas...)

- Worklight Applications have good management for versioning
- However, currently Worklight Adapters **do not**
- Therefore, you should do it **in code** before you reach v1, to prepare yourself for v2:

```
WL.Client.invokeProcedure({
    adapter: "Adapter1",
    procedure: "getStories",
    parameters: [1, "someOtherData"],
});
```

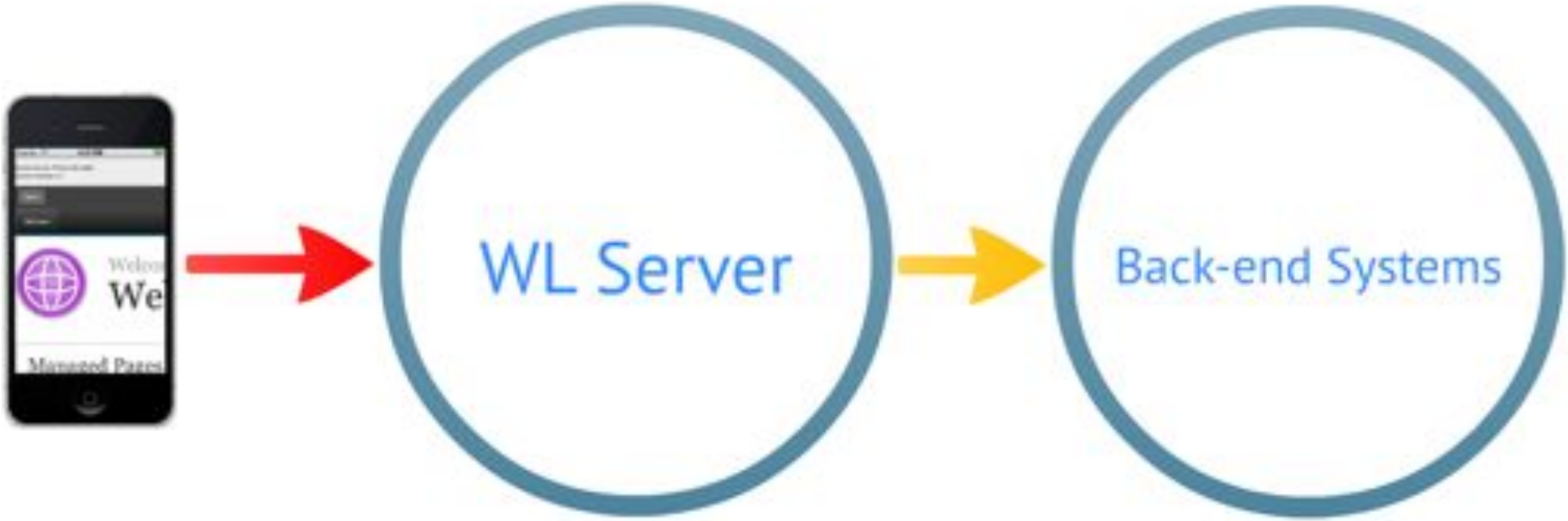
```
WL.Client.invokeProcedure({
    adapter: "Adapter1",
    procedure: "getStories",
    parameters: [1, "someOtherData"],
});
```

```
function getStories(param) {
    if (version === 1) {
        path = getPath(param);

        var input = {
            method : 'get',
            returnedContentType : 'xml',
            path : path,
            transformation : {
                type : 'xslFile',
                xslFile : 'filtered.xsl'
            }
        };

        return WL.Server.invokeHttp(input);
    } else if (version === 2) {
        // ... this code is yet to be written

        // incidentally, it could be factored out into another adapter:

        // var invocationData = {
        //  "adapter" : "StoryV2Adapter",
        //  "procedure" : "getStories",
        //  "parameters" : []
        // };

        // WL.Server.invokeProcedure(invocationData);
    }
}
```

Managing Authentication

- Exploiting standard WL adapter framework outsources authentication to the WL Server 
- App uses WL Challenge Handler to manage client-side login lifecycle; independent of the type of authentication performed.

```
var challengeHandler = WL.Client.createChallengeHandler("WCSAuthenticationRealm");

challengeHandler.isCustomResponse = function(response) {
    // ...
};

challengeHandler.handleChallenge = function(response) {
    var authRequired = response.responseJSON.authStatus;

    if(authRequired === "required") {
        // .. Show our login page
    } else if(authRequired === "complete") {
        // .. Hide our login page; user has successfully
        // logged in.
    }
};
```



Managing Authentication

- Exploiting standard WL adapter framework outsources authentication to the WL Server 
- App uses WL Challenge Handler to manage client-side login lifecycle; independent of the type of authentication performed.

```
var challengeHandler = WL.Client.createChallengeHandler("WCSAuthenticationRealm");

challengeHandler.isCustomResponse = function(response) {
    // ...
};

challengeHandler.handleChallenge = function(response) {
    var authRequired = response.responseJSON.authStatus;

    if(authRequired === "required") {
        // .. Show our login page
    } else if(authRequired === "complete") {
        // .. Hide our login page; user has successfully
        // logged in.
    }
};
```

in lifecycle; independent of the type of
hentication performed.

```
var challengeHandler = WL.Client.createChallengeHandler("WCSAuthenticationRealm");

challengeHandler.isCustomResponse = function(response) {
    // ...
};

challengeHandler.handleChallenge = function(response) {
    var authRequired = response.responseJSON.authStatus;

    if(authRequired === "required") {
        // .. Show our login page
    } else if(authRequired === "complete") {
        // .. Hide our login page; user has successfully
        // logged in.
    }
};
```

How is Server-Side Authentication Managed?

- WL ships with some standard server-side authentication mechanisms (LDAP, LTPA, etc..)
- Many real-world authentication scenarios require more sophisticated authentication
- Worklight provides **Adapter Authentication** for this purpose.

"Custom" Adapter Authentication

- An Authentication Adapter is a specialised HTTP adapter which integrates with the "back-end" login services, etc.

```
function login(username, password) {
    var input = {
        method : 'get',
        returnedContentType : 'json',
        path : '/customer/login?loginId=' + username + "&password=" + password + "&" + apiKey,
    };

    WL.Server.setActiveUser("worklightRealm", { user: username });

    return result;
}
```

- This is often the most complex part of a Worklight implementation, and should be planned and PoCed early.

HTTP adapter which integra

"back-end" login services, et

```
function login(username, password) {
    var input = {
        method : 'get',
        returnedContentType : 'json',
        path : "/customer/login?loginId=" + username + "&password=" + password + "&" + apiKey,
    };

    WL.Server.setActiveUser("WorklightRealm", { user: username });

    return result;

}
```

- This is often the most comp

- Worklight implementation

"Custom" Adapter Authentication

- An Authentication Adapter is a specialised HTTP adapter which integrates with the "back-end" login services, etc.

```
function login(username, password) {
    var input = {
        method : 'get',
        returnedContentType : 'json',
        path : "/customer/login?loginId=" + username + "&password=" + password + "&" + apiKey,
    };

    WL.Server.setActiveUser("WorklightRealm", { user: username });

    return result;
}
```

- This is often the most complex part of a Worklight implementation, and should be planned and PoCed early.

Conclusions

- Worklight is ready for production application construction now - we are doing it with real customers.
- Continuous Integration and Continuous Testing are very powerful for driving up quality and driving down time-to-market even with enterprise complexities.

Agile Mobile Apps for the Enterprise with IBM Worklight