AEM6 Component Development

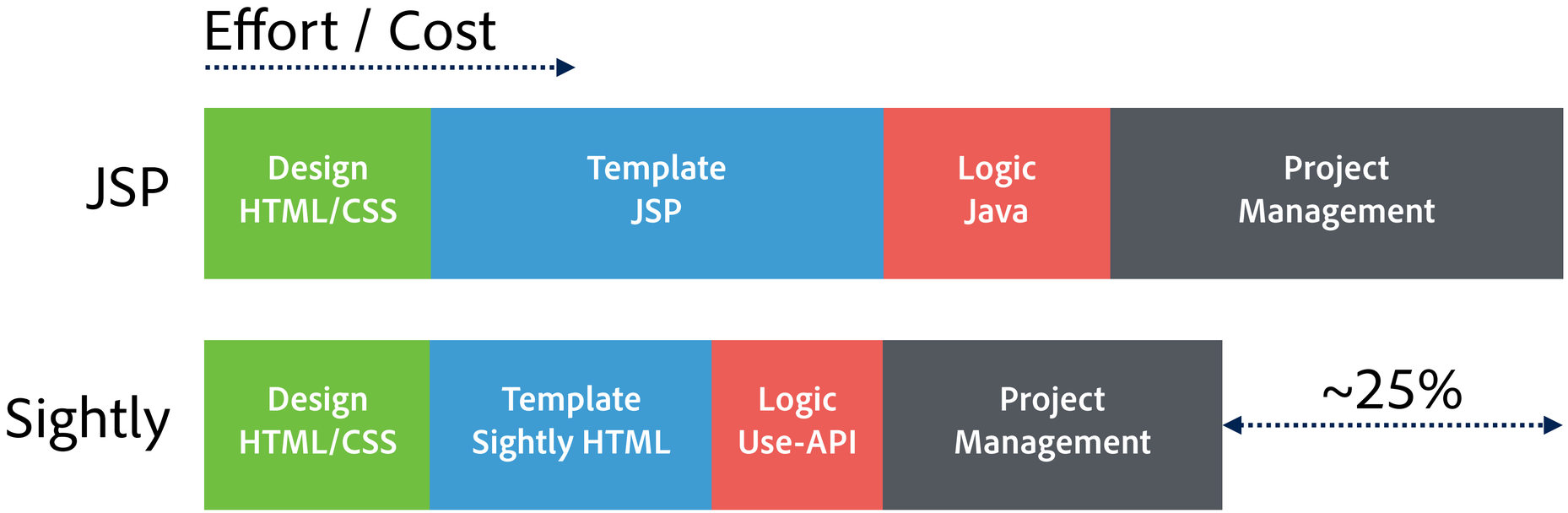
@GabrielWalt, Product Manager

Specification open sourced to GitHub.
Implementation donated to Apache Sling.

Follow @sightlyio on Twitter.
http://docs.adobe.com/docs/en/aem/6-0/develop/sightly.html

Project Efficiency

Adobe.com estimated that it reduced their project costs by about 25%

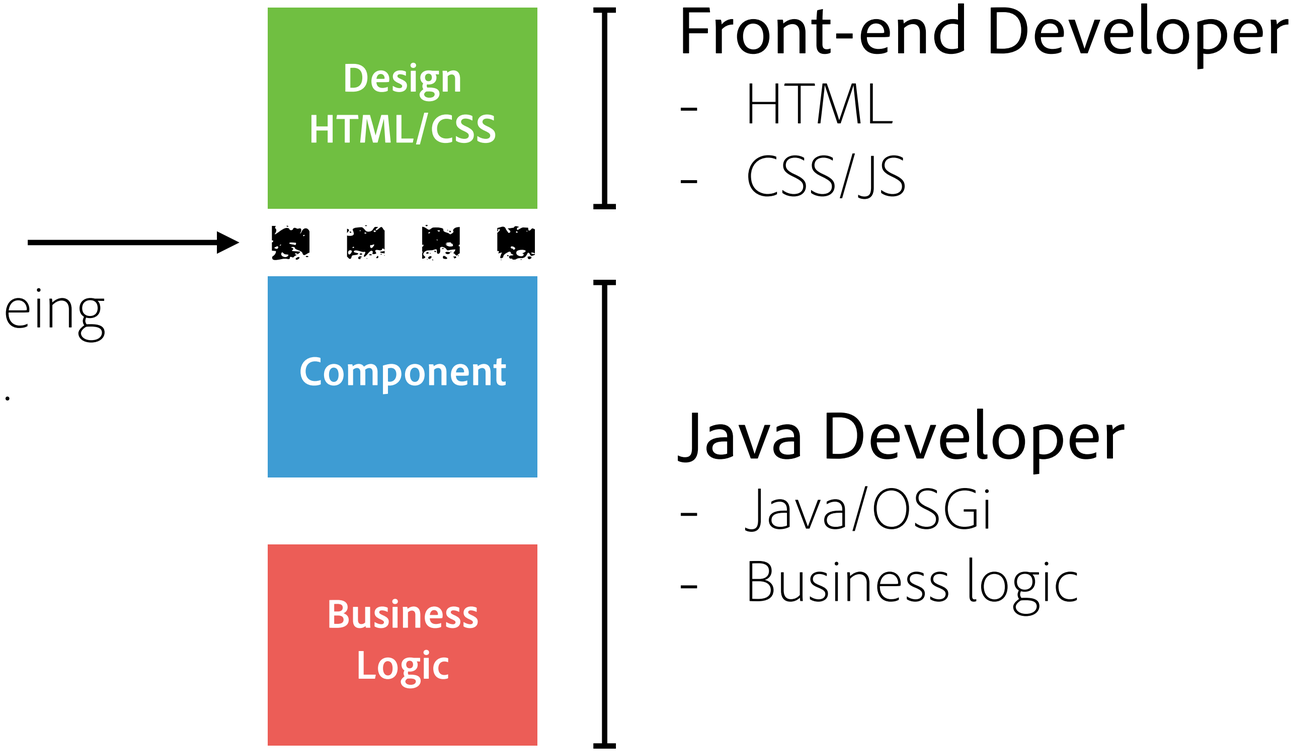


Development Workflow

Improves project efficiency by removing the pain of JSP and Java development

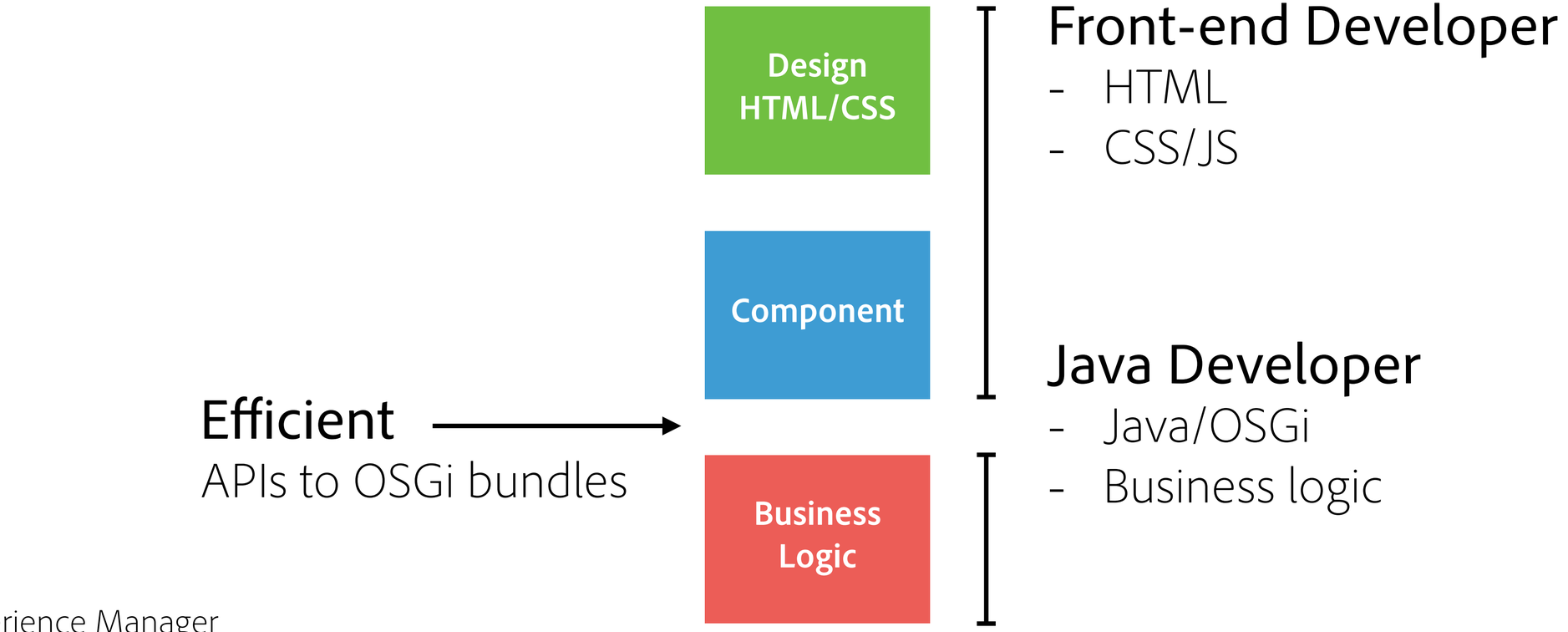
Inefficient

Static HTML being handed over…



Development Workflow

Improves project efficiency by removing the pain of JSP and Java development



Development Workflow

Adobe

Can be edited by front-end devs:

✓ **Client Libraries** (CSS & JS)
× **JSP** (markup & logic)

Component

Development Workflow

Adobe

Can be edited by front-end devs:

- ✓ **Client Libraries** (CSS & JS)
- × ~~**JSP** (markup & logic)~~
- ✓ **HTML markup** (Sightly template)
- ✓ **View logic** (server-side JS, or Java)

Component

Sightly basic example


```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

Sightly basic example

```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

① ② ③

① Automatic contextual HTML escaping and XSS protection of all variables

② Fallback value if property is empty

③ Remove HTML attribute if value is empty

Sightly basic example


```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

Sightly vs JSP

Sightly

```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

JSP – Scriptlets

```
<%@include file="/libs/foundation/global.jsp"%>
<a href="<%= xssAPI.getValidHref(properties.get("link", "#")) %
   String title = properties.get("jcr:title", "");
   if (title.length() > 0) {
      %>title="<%= xssAPI.encodeForHTMLAttr(title) %>"<
   } %>>
   <%= xssAPI.encodeForHTML(properties.get("jcr:des ion", "")) %>
</a>
```

Please try again...

Sightly vs JSP

Sightly

```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

JSP – Expression Language & JSTL

```
<%@include file="/libs/foundation/global.jsp"%>
<a href="${!empty properties.link ? xss:href(properties                    }"
   <c:if test="${!empty properties['jcr:title']}">
      title="${xss:attr(properties['jcr:title'])}"
   </c:if>
>
   ${xss:text(properties['jcr:description'])
</a>
```

No automatic security

Still complex

Sightly vs JSP

Sightly

```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

JSP – Custom Tag Libraries

```
<%@include file="/libs/foundation/global.jsp"%>
<a href="<out:href property='link' default='#'/>"
   <c:if test="${!empty properties['jcr:title']}">
      title="<out:attr property='jcr:title'/>"
   </c:if>
>
   <out:text property='jcr:description'/>
</a>
```

No automatic security

Many tags within tags

Sightly vs JSP

Sightly

```
<a href="${properties.link || '#'}" title="${properties.jcr:title}">
   ${properties.jcr:description}
</a>
```

JSP – TagLibs for whole HTML elements

```
<%@include file="/libs/foundation/global.jsp"%>
<my:link
   urlProperty="link"
   urlDefault="#"
   titleProperty="jcr:title">
   <my:text property="jcr:description"/>
</my:link>
```

What does it really do?

Sightly FTW!

Sightly

```
<a href="${properties.link || '#'}" title="${prope  .j  .tl
   ${properties.jcr:description}
</a>
```

Secure

Readable

Explicit

Building Blocks

Expression Language

```
${properties.myProperty}
```

Block Statements

```
<p data-sly-test="${isVisible}">is visible</p>
```

Expressions

Literals

```
${42}
${true}
${'Hello World’}  ${"Hello World"}
${[1, 2, 3]}      ${[true, 2, 'Hello World']}
```

Variables

```
${myVar}
${properties.propName}
${properties.jcr:title}
${properties['my property']}
${properties[myVar]}
```

Expression Operators

Logical operations

```
${!myVar}
${conditionOne || conditionTwo}
${conditionOne && conditionTwo}
```

Equality / Inequality (only for same types)

```
${varOne == varTwo}    ${varOne != varTwo}
```

Comparison (only for integers)

```
${varOne < varTwo}     ${varOne > varTwo}
${varOne <= varTwo}    ${varOne >= varTwo}
```

Expression Operators

Conditional

```
${myChoice ? varOne : varTwo}
```

Grouping

```
${varOne && (varTwo || varThree)}
```

Expression Options

Options allow to manipulate the result of an expression, or to pass parameters to a block statement.

Everything after the @ is an option

```
${myVar @ optOne, optTwo}
${myVar @ optOne='value', optTwo=[1, 2, 3]}
```

Parametric expression, containing only options

```
${@ optOne='value', optTwo=[1, 2, 3]}
```

Expression Options

String formatting

```
${'Page {0} of {1}' @ format=[current, total]}
```

Internationalization

```
${'Page' @ i18n}
${'Page' @ i18n, hint='Translation Hint'}
${'Page' @ i18n, locale='en-US'}
```

Array Join

```
${['one', 'two'] @ join='; '}
```

Test Statement

Conditionally removes the element and it's content

```
<p data-sly-test="${properties.showText}">text</p>
```

Output

```
<p>text</p>
```

List Statement

Repeats the content for each enumerable property

```
<ol data-sly-list="${currentPage.listChildren}">
    <li>${item.title}</li>
</ol>
```

Output

```
<ol>
    <li>Triangle Page</li>
    <li>Square Page</li>
</ol>
```

Include Statement

Includes the rendering of the indicated template (Sightly, JSP, ESP, etc.)

```
<section data-sly-include="path/to/template.html"></section>
```

Output

```
<section><!-- Result of the rendered resource --></section>
```

Resource Statement

Includes the result of the indicated resource

```
<article data-sly-resource="path/to/resource"></article>
```

Output

```
<article><!-- Result of the rendered resource --></article>
```

Resource Statement Options

Manipulating selectors (selectors, addSelectors, removeSelectors)

```
<article data-sly-resource="${'path/to/resource' @
selectors='mobile'}"></article>
```

Overriding the resourceType

```
<article data-sly-resource="${'path/to/resource' @
resourceType='my/resource/type'}"></article>
```

Changing WCM mode

```
<article data-sly-resource="${'path/to/resource' @
wcmmode='disabled'}"></article>
```

Unwrap Statement

Removes the host element while retaining its content

```
<article data-sly-resource="path/to/resource"
         data-sly-unwrap></article>
```

Output

```
<!-- Result of the rendered resource -->
```

Use unwrap only when there's no other way to write your template, to make it correspond as much as possible to the output.

Text, Attr & Elem Statements

Replaces the content, attribute or element name

```
<div class="active" title="Lorem ipsum"
     data-sly-element="${elementName}"
     data-sly-attribute.title="${title}"
     data-sly-text="${content}">Lorem ipsum</div>
```

Output

```
<span class="active" title="Hi!">Hello World</span>
```

Use element and attribute with care as they allow to define parts of the markup from the logic, which can lessen the separation of concerns.

Use Statement

Initializes a helper object

```
<div data-sly-use.nav="navigation.js">${nav.foo}</div>
```

Output

```
<div>Hello World</div>
```

Use data-sly-use when data preparation is needed, but avoid to use it as a wrapper that doesn't do anything additionally. Prefer to access the variables directly from within the template when possible.

Server-side JavaScript logic

```
<!--/* template.html */-->
<div data-sly-use.nav="navigation.js">${nav.foo}</div>

<!--/* navigation.js */-->
use(function () {
    return {
        foo: "Hello World"
    };
});
```

Java logic

```
<!--/* template.html */-->
<div data-sly-use.nav="Navigation">${nav.foo}</div>

<!--/* Navigation.java */-->
package apps.site_name.component_name;
import com.adobe.cq.sighty.WCMUse;

public class Navigation extends WCMUse {
    private String foo;

    @Override
    public void activate() throws Exception {
        foo = "Hello World";
    }

    public String getFoo() {
        return foo;
    }
}
```

Template & Call Statement

```
<!--/* template.html */-->
<template data-sly-template.one="${@ class, text}">
   <span class="${class}">${text}</span>
</template>

<!--/* other-file.html */-->
<div data-sly-use.tmpl="template.html"
     data-sly-call="${tmpl.one @ class='example',
                                 text='Hi!'}"></div>
```

Output

```
<div><span class="example">Hi!</span></div>
```

Display Context Option

The context option offers control over escaping and XSS protection.

Allowing some HTML markup (filtering out scripts)

```
<div>${properties.jcr:description @ context='html'}</div>
```

Adding URI validation protection to other attributes than `src` or `href`

```
<p data-link="${link @ context='uri'}">text</p>
```

Display Context Option

```
<a href="${myLink}" title="${myTitle}">${myContent}</a>
<script> var foo = "${myVar @ context='scriptString'}"; </string>
<style> a { font-family: "${myFont @ context='styleString'}"; } </style>
```

Most useful contexts and what they do:

safer ↑

`number`	`XSSAPI.getValidNumber`
`uri`	`XSSAPI.getValidHref` (default for `src` and `href` attributes)
`attribute`	`XSSAPI.encodeForHTMLAttribute` (default for other attributes)
`text`	`XSSAPI.encodeForHTML` (default for element content)
`scriptString`	`XSSAPI.encodeForJSString`
`styleString`	`XSSAPI.encodeForCSSString`
`html`	`XSSAPI.filterHTML`
`unsafe`	disables all protection, use at your own risk.

Display Context Option

```
<a href="${myLink}" title="${myTitle}">${myContent}</a>
<script> var foo = "${myVar @ context='scriptString'}"; </string>
<style> a { font-family: "${myFont @ context='styleString'}"; } </style>
```

Preferred method for each context

- `src` and `href` attributes: `number, uri, attribute, unsafe`
- other attributes: `number, uri, attribute, unsafe`
- element content: `number, text, html, unsafe`
- JS scripts (*): `number, uri, scriptString, unsafe`
- CSS styles (*): `number, uri, styleString, unsafe`

(*) An explicit context is required for script and style contexts.
Don't set the context manually unless you understand what you are doing.

Sightly FAQ

What does Sightly enable that isn't possible with JSP?

- Systematic state-of-the-art protection against cross-site scripting injection.
- Possibility for front-end developers to easily participate on AEM projects.
- Flexible and powerful templating and view logic binding features.
- Tailored to the Sling use-case.

Will JSP go away?

- As of today, there are no plans for that.

IDE & Developer Mode

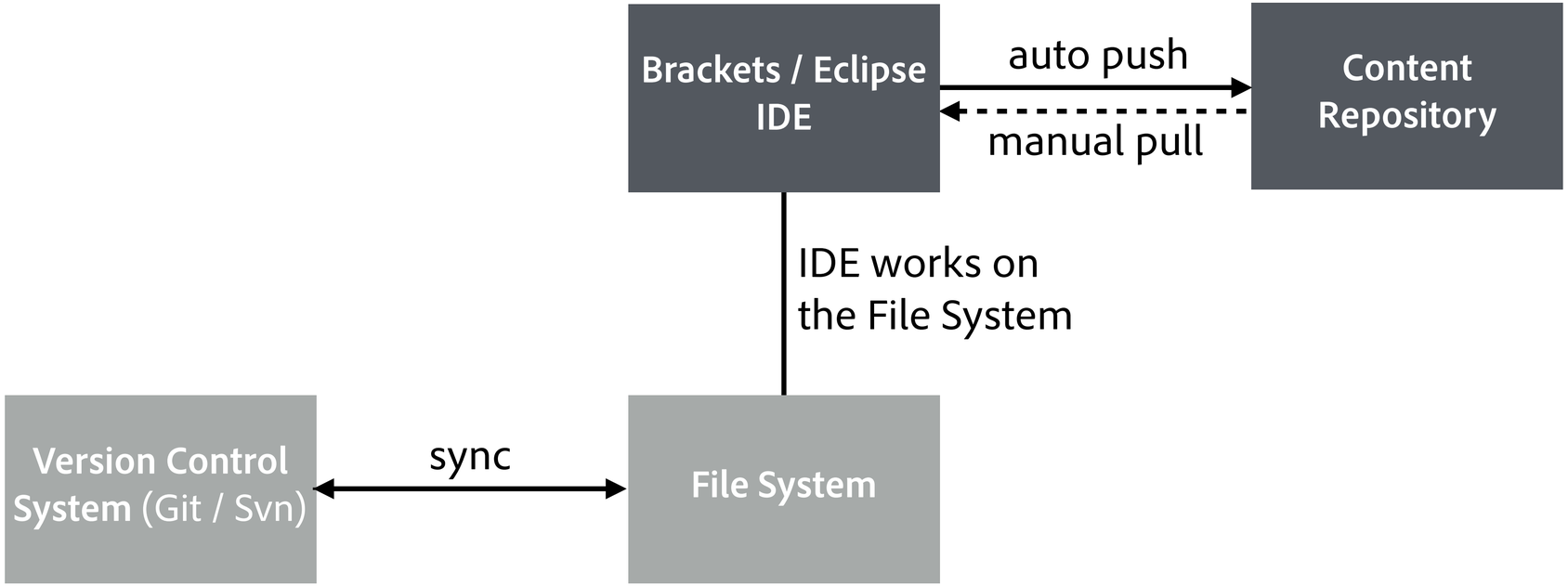
- Improve learning curve and efficiency of developers
- An IDE plugin for each developer role

Brackets plugin
for the Front-End developers
http://docs.adobe.com/docs/en/dev-tools/sightly-brackets.html

Eclipse plugin
for the Java developers
http://docs.adobe.com/docs/en/dev-tools/aem-eclipse.html

IDE Sync

Work on file system + transparent sync & content editing



Resources

Sightly

- Documentation
- Specification
- Sightly AEM Page Example *(requires instance on localhost:4502)*
- TodoMVC Example

Tools

- Live Sightly execution environment
- Sightly Brackets extension
- AEM Developer Tools for Eclipse
- AEM Developer Mode

Adobe