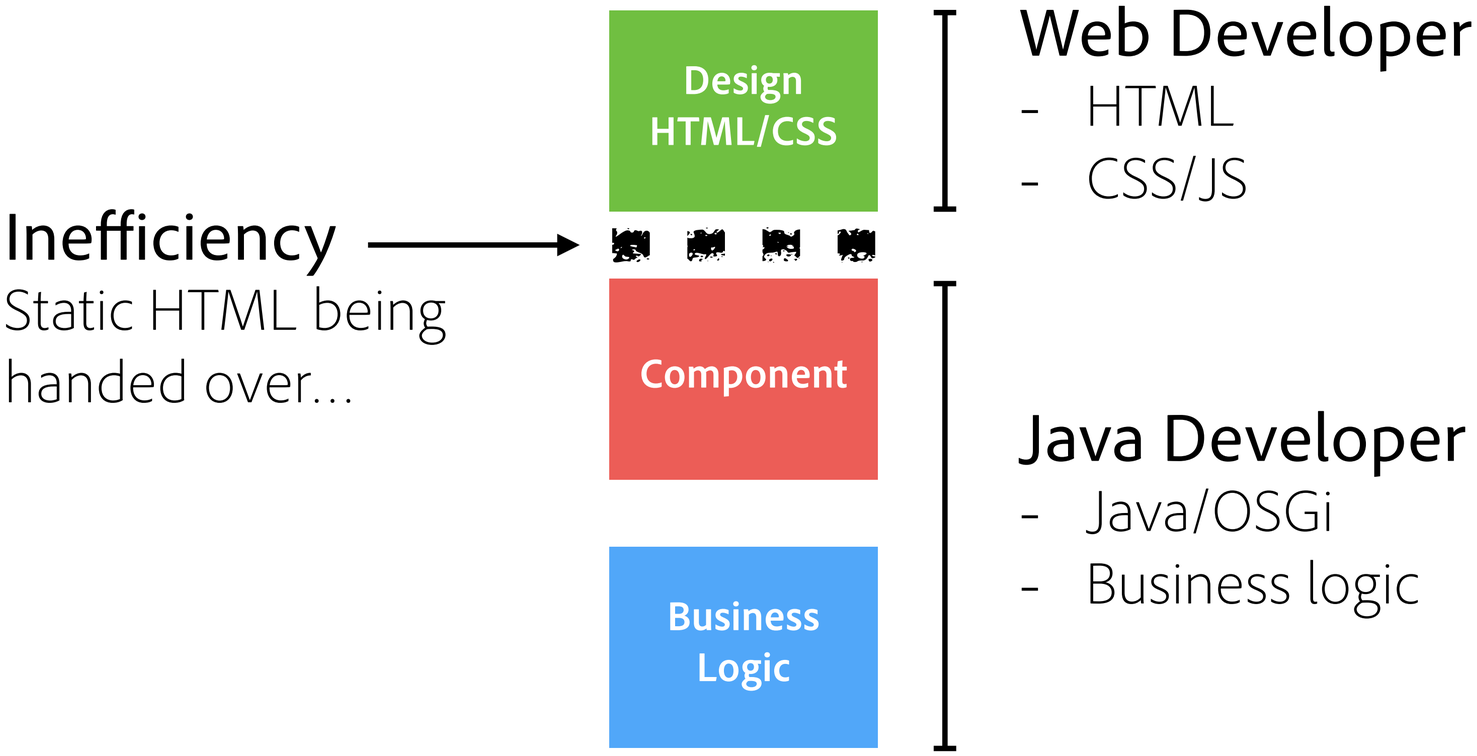
Efficiently Build Reusable Components

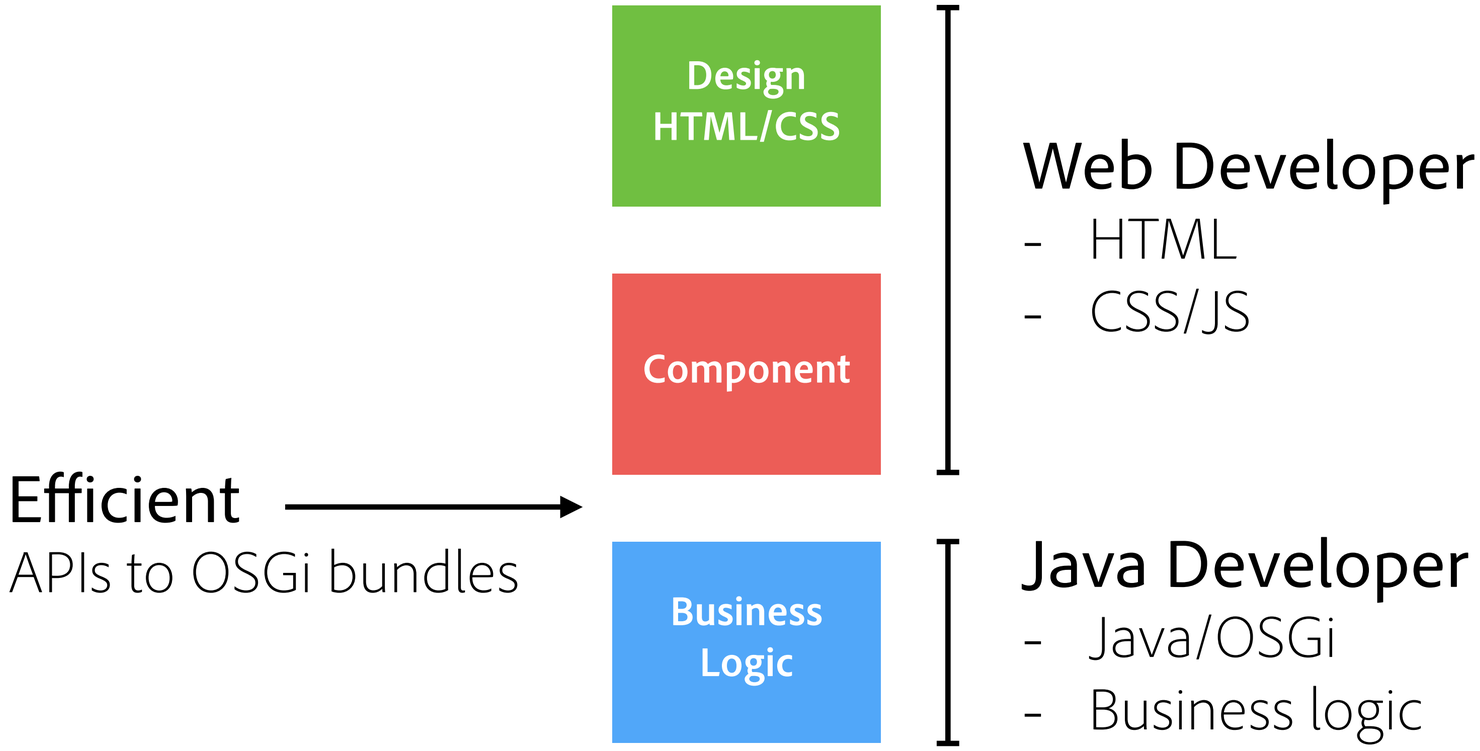
@GabrielWalt, Product Manager, Web Experience Management

- Efficient Component Development
- Component Reusability

Development Workflow



Development Workflow



Development Workflow

Component

Can be edited by Web Developer:

× **JSP** (HTML markup & component logic)

✓ **Client Libraries** (CSS & JS)

Development Workflow

Component

Can be edited by Web Developer:

- ✓ **HTML markup** (Sightly template)
- ✓ **Component logic** (server-side JS)
- × ~~**JSP** (HTML markup & component logic)~~
- ✓ **Client Libraries** (CSS & JS)

Development Workflow

Sightly vs JSP

- HTML encoding + XSS protection
- Default values for empty strings
- Remove empty HTML attributes

Sightly

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

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:description", "")) %>
</a>
```

Sightly Building Blocks

Expression Language

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

Block Statements

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

Use-API

```
<p data-sly-use.obj="script.js">${obj.text}</p>
```

Development Workflow

Two developer roles

Component

Web Developer
- HTML
- CSS/JS

Business Logic

Java Developer
- Java/OSGi
- Business logic

Development Workflow

An IDE plugin for each developer role

Brackets plugin

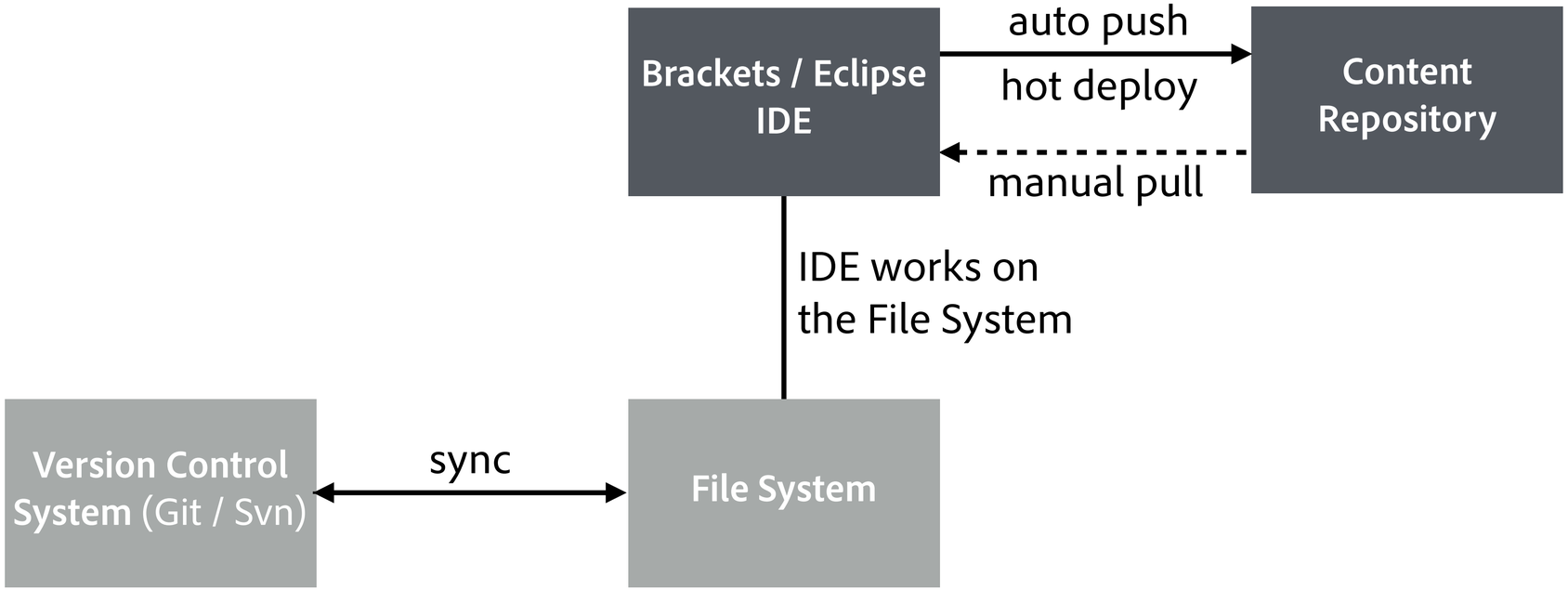
- Sightly code completion & syntax highlighting
- Content nodes & properties manipulation

Eclipse plugin

- Content nodes & properties manipulation
- Bundle development & hot deployment

IDE Sync

Work on file system + transparent sync & content editing



Models of Reusable Components

- **Direct Reuse**
  - Mostly primitives
  - Example: Foundation Text, Image, or Video component
- **Direct Reuse w/ Context**
  - Example: Foundation Column Control component
- **Extensible**
  - Designed for extension per project
  - Example: Foundation List component
- **Sample**
  - Pattern library
  - Example: Geometrixx Tabs component

Conditions for Reusability

- **Texts:** Customizable and/or Internationalized
- **Markup:** Minimal and Semantic
- **Behavior:** Configurable with smart defaults

Patterns for Extensibility

- **Extend dialogs**
  - TouchUI dialogs can use the powerful Sling Resource Merger.
  - ClassicUI dialogs have the cqinclude xtype to split big dialogs into overridable pieces.
- **Overlay markup templates**
  - Split long markup chunks into overridable pieces.
- **Extend component logic** (with the Sightly Use-API)
  - JavaScript: The use() call's dependency loader allows to reuse the result from existing components' JS logic.
  - Java: Inheritance allows to extend the logic of components.

Anti-patterns

- No separation between markup and logic.
- Dependencies on a particular CSS or JS framework.
- Login/Registration–related components.
- Lack of Governance.
- Overly Aggressive Goals.

Resources

Sightly Documentation

http://dev.day.com/content/docs/en/aem/6-0/develop/sightly.html

Sling Resource Merger

http://docs.adobe.com/docs/en/aem/6-0/develop/platform/overlays.html

Experience Delivers blog posts

http://experiencedelivers.adobe.com/