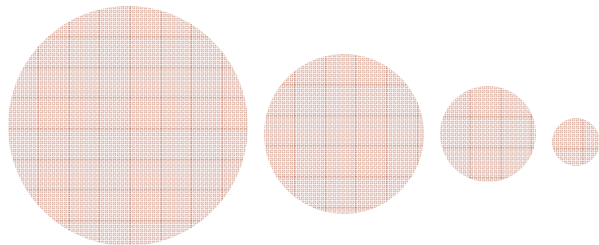


JSF Vs Tapestry Vs Wicket – A Perspective



IndicThreads.com Conference On Java Technology
26th & 27 Oct. 2007 Pune, India



Karthik Gurusurthy,
Author – Pro Wicket, Apress Publications,
Architect, Symcon Global Technologies Inc

The Java Ecosystem

‘ When depressed, Women consume chocolates & go shopping while men invade other countries’

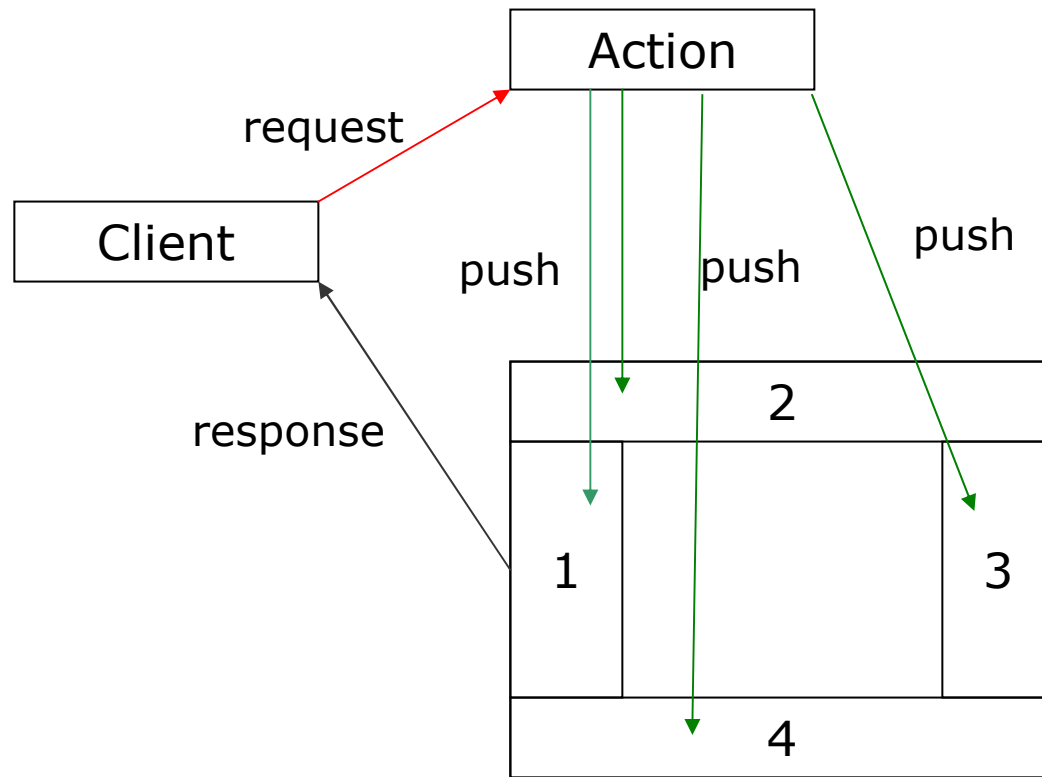
- An old proverb

‘ When depressed, **Women consume chocolates & go shopping** while men create “yet another open source java web framework” ’

- Karthik Gurumurthy ;-)
(adopted from an old proverb)



Model2 (Struts, Spring MVC)



Hello.jsp



Component Oriented (Wicket, JSF, Tapestry..)



Where Tapestry seems headed

- Current stable version of Tapestry is 4.1.2
- The development version that is being worked upon in Tapestry 5.0 (T5)
- T5 is a rewrite from the ground-up
 - NOT backward compatible with 4.1.2
- Tapestry 5.0 has not had a 'beta' for quite *sometime* now
 - No Ajax support yet for e.g
- T5 is likely to be very innovative (if &) when it is finally released
- *Hope* to discuss T5 in IndicThreads Conf 2008!



What Works in favor of JSF?

- Java EE Standard
 - Easy to convince upper management (Yours/ Client's)
 - Lots of third-part libraries, quite a few implementations
- Relatively easier to find skills in the market
- Great Vendor IDE support
- More Productive than Struts.
- Ability to render differently based on the client type (Browser, CellPhone e.t.c)
- 'Relatively' Easier for 'existing' java web developers to relate to
 - JSP-like (taglibs) and Struts-like design (externalized navigation e.t.c)



What Works for Apache Wicket

- Off-late, Apache banner! And a great community!
- Very productive once you get a hang of Wicket ‘Models’
- The ease of development of Custom Components might actually bring tears (of joy) to your eyes
- ‘When the going gets tough, the tough gets going’
 - The Programming Model shines through with complex UI Requirements
- Can carry forward your desktop-based application development skills to the Web
- No XML Configuration (Ah - No Annotations either ☺)
- Works with plain Java and HTML/CSS
 - HTML Designers get to use their tool !
 - A total reliance on Java means automatic refactoring support, static typing and the like



What does NOT Work (Wicket)

- ‘Everything in Java’ could be annoying sometimes (for a while that is ☺)
 - Java does reload classes as long as there aren’t too many structural changes to the classes though
- Absence of ‘declarative programming’ style mandates good level of familiarity with the framework Java API
- Session heavy despite several design considerations to keep sessions lighter
- The “core team” might not support some of the high end components like Dojo for e.g
- Relatively new and hence difficult to sell to management



What does NOT Work (JSF)

- Java EE Standard – Too many implementations that don't seem to interoperate? Which one should I Choose?
- Complex UI requirements might bring tears to the eyes (of a different kind ☺)
 - A “Component Based” framework that does not help you to easily create custom Components?
- Component based frameworks store the component tree in the memory and might not perform as fast as stateless request based Model2 frameworks.
- “Type based Java programming Vs ‘Strings’ programming” (EL, XML)
 - Where are my types / Strongly typed arguments , Classes/Methods , Refactoring Support?
- JSF Pages look like an ocean of tags!
 - Code generated through “drag & drop” looks really scary sometimes



The Template (JSF)

```
<h:form>
  <h:messages/>
  <h:inputText value="#{userBackingBean.userId}"
    required="true"/>
  <h:inputSecret
    value="#{userBackingBean.password}"
    required="true"/>
  <h:commandButton action="#{login}"/>
</h:form>
```



The Backing Bean(JSF)

```
Class UserBackingBean{  
    String userId;  
    String password;  
    //getters and setters  
    public void login(){  
        getUserId();  
        getPassword();  
    }  
}
```

Add an entry in faces-config.xml for this bean



The Template (Tapestry)

```
<form jwcid="@Form" listener="listener:login">  
  User Id: <input type="text" jwcid="@TextField" value="ognl:userId"/>  
  
  Password: <input type="text" jwcid="@PasswordTextField"  
    value="ognl:password"/>  
  
  <input type="submit" value="Login"/>  
</form>
```



The Backing Page (Tapestry)

```
public abstract class LoginPage extends BasePage{  
    public abstract String getUserId( );  
    public abstract String getPassword();  
  
    public void login(IRequestCycle cycle) {  
        if(authenticate(getUserId(), getPassword())){  
            WelcomePage page = getWelcomePage():  
            page.setUser(///);  
            cycle.activate(page);  
        }  
    }  
  
    @InjectPage("welcome")  
    public abstract WelcomePage getWelcomePage();  
}
```

Tapestry subclasses your page at runtime and provides implementation of abstract methods

Page-instances are pooled and hence instance-variables like these need to be cleaned up after every request

The Template (Wicket)

```
<form wicket:id="form">
```

```
  User Id: <input type="text" wicket:id="userid" />
```

```
  Password: <input type="text" wicket:id="password" />
```

```
    <input type="submit" value="Login"/>
```

```
</form>
```

- Login.html



The Backing Page (Wicket)

```
public class Login  
    public LoginPag  
        User user = n  
        Form form =  
        @Override  
        public void c  
            setRespo  
        }  
    }  
  
    form.setMode  
    form.add(new  
    form.add(new  
    add(form);
```

WelcomePage.html

```
Welcome <span wicket:id="user"></span>
```

WelcomePage.java

```
public class WelcomePage extends WebPage{  
    public WelcomePage(User user){  
        add(new Label("user",user.getUserId( ));  
    }  
}
```



Observations

- Model and Controller
 - A POJO can act as both the Model and the Controller in JSF
 - ‘n-1’ Mapping between View and Controller
 - But the spec-leads seem to encourage 1-1 mapping (JSF Complete Reference)
 - Backing beans
 - Wicket (& Tapestry) requires a Page class for every view
 - The Page class typically extends from a Framework class like any other component
 - 1-1 mapping between the View and Controller is a ‘requirement’



Observations

- **The Object Oriented 'Feel'**

- The Programming model (Tapestry and JSF) typically result in a backing bean / page class with a static method-like feel to it.
 - The components listeners are bound to these methods in the template.
- Wicket on the other hand embraces the traditional Object Oriented flavor.
 - Well suited for developers from GUI Application programming background
 - Components are “instantiated” by the developer and Components have listener methods



Observations

faces-config.xml and Managed beans

- The in-direction offered by JSF in the form of faces-config.xml is quite distracting sometimes.
 - The String based look up makes it only worse
- Model binding in Wicket & Tapestry is 'explicit'.
 - Any bean accessed in a Tapestry template needs to be made available in the corresponding Page class
 - Wicket is all about Java and Static type checking



Observations

- Learning Curve
 - Transition from request/action “push” based Model2 Frameworks to Component based “pull” development.
 - Wicket’s obsession with Java might feel a little distracting (boiler plate code) and seem counter productive (for a while that is 😊)
 - JSF is natural for developers that come from Java Web Development background (mainly Struts).
 - Same old tag library soup, JSP EL, externalized “string” based navigation JSF’s “another level of in-direction” syndrome for just about anything leads to more artifacts
 - JBoss Seam fixes it to an extent



Observations

- Learning Curve
 - Wicket is natural for Desktop Application Developers
 - Wicket favors convention over configuration in many of the design choices
 - **Not** Inspired by RoR
 - Leads to fewer artifacts
 - Page navigation expressed in Java with Strong typing
 - Tapestry 4.1 doesn't require a page specification file any longer
 - Needs annotations now in the Page class
 - Page navigation can be both "String" based and Java based
 - If Java, supports Strong typing



Observations

- Learning Curve
 - Tapestry is “known” for its steep learning curve
 - But once you get a hang of it , seems quite productive
 - Requires knowledge of its own IOC container – Hivemind
 - Tapestry 4.1 doesn't require a page specification file any longer
 - Needs annotations now in the Page class
 - Page navigation can be both “String” based and Java based
 - If Java, supports Strong typing



Observations

- 'Previewable' templates
 - Wicket and Tapestry use HTML templates that are previewable on the browser
 - JSF views are NOT previewable on the browser for obvious reasons
 - If you require previewability, use facelets
 - Facelets is non-standard
 - » But commonly viewed as the only practical way to make JSF work!



Observations

- Tapestry Vs Wicket
 - Wicket innovates with backward compatibility in mind
 - Tapestry employs high end technology and every version is typically a re-write
 - A DSL based on quite a few annotations
 - Javaassist for bytecode manipulation
 - Hivemind IOC
 - Tapestry Pages are pooled and shared (but thread-safe) between requests hence performance is no:1 criteria
 - Wicket is about ease of development
 - A page instance is stored in session and belongs to the user unlike pooled tapestry pages
 - Wicket mailing list support is probably one of the best in OSS
 - Wicket community is extremely 'user-driven'
 - If you ask for something legitimate, the support might be 'checked-in' in a day / two!

Ajax (JSF)

- The “standard” spec does not talk about Ajax support
 - RichFaces, jmaki, ICEFaces e.t.c
 - Ajax w/o writing a single line of javascript code
 - Mix them together and experience the hell
 - A Component market ? !
- Recommended approach : Pick one implementation and stick to it



Ajax (Wicket)

- Wicket ships with a powerful baked-in Ajax support
 - Ajax is modeled as a ‘behavior’ exhibited by a Component and can be added dynamically at runtime.
 - Can have “Ajax-only” components
 - “Ajax fallback” behavior
- Wicket-Ajax is a very good example of ‘scale’
- Multiple Components can be ‘repainted’ in response to an Ajax request
 - Can execute any arbitrary “javascript” as well in response to an Ajax request
- Ajax Validation is standard form of validation
- Ajax w/o requiring you to write single line of javascript
- Dojo, Scriptaculous integration maintained as ‘*wicket-stuff*’ projects



Bookmarkability

- JSF
 - Seems ‘post’ obsessed
 - Bookmarkability is an issue in JSF 1.2
 - JSF 2.0 will hopefully address this issue
 - Tapestry – supports URLs
- Wicket
 - pages can be ‘mounted’ and ‘bookmarkable’



Validation Support

Client side Validation support

- JSF (spec) does not support client-side validation
- Tapestry supports client-side validation
- Wicket's client-side validation support was deprecated in favor of Ajax validation

Customizing Validation Messages

- **(JSF)** Custom validation messages can be supplied against appropriate message keys. For e.g : keys used internally
 - **javax.faces.component.UIInput.REQUIRED**
 - **javax.faces.validator.LengthValidator.MINIMUM**
- Both Wicket and Tapestry also allow you to modify the validation messages easily



Specifying Object Scopes (JSF)

Bean Scopes in faces-config.xml

```
<faces-config>
  <managed-bean>
    <managed-bean-name>user</managed-bean-name>
    <managed-bean-class>com.sgt.User</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

“session”, “request” and “application” are the possible scopes

- Declarative & Flexible management of bean scopes
- Loss of type
- faces-config.xml clutter



Specifying Object Scopes (Wicket)

```
public class MyAppSession extends WebSession{  
    private User user;  
  
    public User getUser(){ return this.user;}  
}
```

“Session” scoped Data
goes here

```
public MyApplication extends WebApplication{  
    private List<Data> lookUpData;  
  
    public List<Data> getLookUpData() { return this.lookUpData;}  
}
```

“Application” scoped Data
goes here

- **Wicket’s WebSession class is a nice façade over Http Session**
- **One instance of WebApplication class exists per web application**
- **Wicket requires a custom WebApplication class be specified in web.xml**
- **Custom WebSession class is configured through factory class**

Accessing Session/Application Objects (JSF)

Accessing the 'userId' attribute of 'UserBean' stored in the FacesContext

```
FacesContext context = FacesContext.getCurrentInstance( );  
ELContext elContext = context.getELContext( );  
Application application = context.getApplication( );  
String userId = (String) application.  
    evaluateValueExpressionGet(context, "#{userBean.userId}", "String.class");
```

Invoking Methods on Managed Beans (JSF)

Invoking a method

```
FacesContext context = FacesContext.getCurrentInstance( );
ELContext elContext = context.getELContext( );
MethodExpression me = expressionFactory
    .createMethodExpression((elContext,
    “#{UserBean.addPermisson.class,Void.TYPE, new Class[]{}String.class}”);
try{
    result = me.invoke(elContext, new Object[]{....}
}
```

Is this Java Programming? ☹

Accessing Session/Application Objects (Wicket)

```
public Class MyBasePage extends WebPage{  
    public MyAppSession getMySession(){  
        return (MyAppSession) getSession();  
    }  
}
```

“Layered” Pages

```
public MyApplication getMyApplication(){  
    return (MyApplication) getApplication();  
}  
}
```

Functionality Common
to all pages

```
public Class EditProductPage extends MyBasePage  
public EditProductPage () {  
    User user = getMySession().getUser();  
    List<Data> data = getMyApplication().getLookUpData();  
}  
}
```

Strongly typed interaction with the
Session and Application state



Validation Messages (JSF)

Default Validation Messages could seem ugly

```
<h:inputText id="loginid" value="#{login.loginId}" required="true" />  
Error: j_id_id13:loginid: Validation Error: Value is required.
```

```
<h:inputText id="loginid" value="#{login.loginId}" required="true" label="#{msgs.name}"/>  
#message bundle entry  
name=Login
```

Error: Login: Validation Error: Value is required.

Validation Messages Could be customized though

```
#message bundle entry  
reqMessage={0} is required  
name=Login
```

```
<h:inputText id="loginid" value="#{login.loginId}" required="true" label="#{msgs.name}"  
requiredMessage="#{msgs.reqMessage}" />
```

Validation Messages (Wicket)

Default Validation Messages look OK

Template : `<input type="text" wicket:id="loginid"/>`

Page Class : `add(new TextField("loginid").setRequired(true));`

Error: *'loginId' is required.*

#page.properties /panel.properties / CustomWebApplicaton.properties entry

loginId=Name

Error: *'Name' is required.*

Interpolation works!

Validation Could be customized

#message bundle entry

loginid.RequiredValidator=Field {label} is required

loginId=Name

Error: *Field 'Name' is required.*

Chaining Validators (JSF)

Specifying Multiple Validators for a Component

```
<h:inputText id="card" value="#{cardService.card}" required="true">  
  <f:validateLength minimum="13"/>  
</h:inputText>
```

JSF Offers Several ways of binding Validators to components:

```
<h:inputText id="card" value="#{payment.card}" required="true">  
  <f:validator validatorId="javax.faces.validator.LengthValidator">  
    <f:attribute name="minimum" value="13"/>  
  </f:validator>  
</h:inputText>
```

Binding validation to a backing bean method

```
<h:inputText id="card" value="#{payment.card}" validator="#{card.validateCard}">
```

```
public void validateCard(FacesContext ctx,UIComponent comp,Object value){ }
```

Chaining Validators (Wicket)

Adding Validators to a Wicket Component through add(IValidator) method

Template : `<input type="text" wicket:id="card"/>`

Page Class : `add(new TextField("card"))`

`.setRequired(true).add(NumberValidator.minimum(13));`

Wicket supports only one (and common-sense) way of associating validators with Components

- All Wicket Validators need to implement wicket's **IValidator** interface
- Wicket Form components typically extend from **FormComponent**
 - **FormComponent.add(IValidator validator)**



Converter (JSF)

JSF Converters implement Converter interface

```
public interface Converter{  
    Object getAsObject(FacesContext context, UIComponent component, String newValue);  
    String getAsString(FacesContext context, UIComponent component, Object value);  
}
```

Implicit conversions for primitive types

Registering Converters

```
<converter>  
  <converter-id>com.sgt.jsf.PhoneConverter</converter-id>  
  <converter-class>com.sgt.PhoneConverter</converter-class>  
</converter>
```

'id' that identifies the converter

Converter implementation class



Converter (JSF)

Explicitly Specifying Converters if converter NOT registered

```
<h:inputText id="phone" value="#{emp.phone}" converter="com.sgt.PhoneConverter"/>
```

Using Method binding to specify converter

```
public class Employee {  
    public Converter getConverter() {  
        return new Converter() {  
            public Object getAsObject(FacesContext context, UIComponent  
                component, String newValue) throws ConverterException { ... }  
            public String getAsString(FacesContext context, UIComponent component,  
                Object value) throws ConverterException { ... }  
        };  
    }  
}
```

```
<h:inputText id="phone" value="#{emp.phone}" converter="#{emp.converter}"/>
```

Converter (Wicket)

Wicket Converters implement IConverter interface

```
public interface IConverter{  
    Object convertToObject(String value, Locale locale);  
    String convertToString(Object value, Locale locale);  
}
```

Implicit conversions for primitive types



Converter (Wicket)

Registering Converters Globally at the time of Initialization

```
public class MyApplication extends WebApplication {  
  
    //Wicket's Application class calls this factory-method at start-up time  
  
    @Override  
    protected IConverterLocator newConverterLocator() {  
        // Get the default IConverterLocator  
        IConverterLocator old = super.newConverterLocator();  
        // Decorate the default locator with your custom built one  
        CustomConverterLocator newConverterLocator = new  
            CustomConverterLocator(old);  
        newConverterLocator.set(PhoneNumber.class,  
            PhoneNumberConverter.INSTANCE);  
        return newConverterLocator;  
    }  
}
```



Converter (Wicket)

```
public class CustomConverterLocator implements IConverterLocator {
    private IConverterLocator old;
    private Map<Class, IConverter> classToConvMap = new HashMap<Class, IConverter>();

    public CustomConverterLocator(IConverterLocator old) {
        this.old = old;
    }

    public IConverter getConverter(Class type) {
        IConverter converter = classToConvMap.get(type);
        if (converter != null) {
            return converter;
        } else {
            return old.getConverter(type);
        }
    }

    public void set(Class clazz, IConverter converter) {
        classToConvMap.put(clazz, converter);
    }
}
```

A CustomConverterLocator
implemented like a “decorator”

Converter (Wicket)

Explicitly Specifying Converters if converter NOT registered:

```
<input type="text" wicket:id="phone" />
```

Wicket allows Components to specify their Converters:

```
add(new TextField("phone"){  
    @Override  
    public IConverter getConverter(Class type){  
        return PhoneConverter.INSTANCE;  
    }  
});
```



Converter (Wicket)

If you are required to display Phone Numbers in more than one page...

```
public Class PhoneField extends TextField{
    public PhoneField(String id,IModel model){
        super(id,model,PhoneNumber.class);
    }

    @Override
    public IConverter getConverter(Class type){
        return PhoneConverter.INSTANCE;
    }
}
```

HTML: `<input type="text" wicket:id="phone"/>`

Java: `add (new PhoneField("phone", new PropertyModel(employee,"phone"));`



Duplicate Post Problem

- Solution – “Redirect after post”
 - This is “**default**” Wicket behavior
 - In JSF it isn't! – Why?
 - In Tapestry, you do it by throwing an exception!
 - JSF requires you to configure ‘explicitly’ in faces-config.xml

```
<navigation-case>  
  <from-outcome>success</from-outcome>  
  <to-view-id>/success.jsp</to-view-id>  
  <redirect/>  
</navigation-case>
```



Custom Component Development

- JSF * (Painful – Facelets seems to ease that a bit)
- Tapestry * * * * (Pioneered the ease of Custom Component Dev)
- Wicket * * * * * (Wicket just picks up from Tapestry and takes it to a different level)
 - Place all resources (HTML/ CSS/ JS/ Images) in the same location as the Java class.
 - Header Contribution is a breeze
 - Jar them up and place it in classpath and get eclipse to *import* the custom Component class.



Localization Support (JSF)

- Resource Bundles are declared and accessed in every page
 - `<f:loadBundle key=“`
- In JSF 1.2, they can be declared globally in `faces-config.xml`



Localization Support (Wicket)

- All Pages typically have a resource bundle by the same name in the same location
- Sophisticated Resource Bundle Lookup
- L10n in pages through `<wicket:message key="name">`
- You extend from Framework classes (WebPage, Panel, FormComponent e.t.c)
 - The Localizer is just a method call away -
 - `getLocalizer.getString(Component scope, String resourceKey, String defaultMessage);`

Tools

- JSF tool support is way ahead
 - Well it was created for that!
- Wicket has netbeans. IntelliJ and eclipse support
 - Not used extensively though
- Tapestry “had” a great eclipse-plugin – Spindle
 - Does not exist for 4.1



Spring Integration (JSF)

- **Spring's DelegatingVariableResolver makes JSF – Spring Integration transparent**
 - But doesn't seem to handle namespace collisions
 - Some issues...



Spring Integration (Wicket)

Wicket-Spring integration through @SpringBean annotation

Reference to Model beans are explicit

Wicket Components can access Spring Beans through the @SpringBean annotation

```
public Class ProductCreatePage extends WebPage{

    @SpringBean private ProductService productService;

    private Product product = new Product();

    public ProductCreatePage (){
        add(new TextField("name", new PropertyModel(product," name"));
        //..
        add(new Button("save"){
            public void onSubmit(){
                productService.add(product);
            }
        }
    }
}
```



Spring Integration (Wicket)

- Wicket-Spring Integration through @SpringBean annotation
 - Or directly look up from Application class
 - Wicket is an un-managed framework
 - Generic Wicket-Injection Framework
 - Spring and Google Guice integration



Spring Integration (Tapestry)

Tapestry-spring sub project

```
Public abstract Class ProductCreatePage extends WebPage{
```

@InjectSpring

```
public abstract ProductService getProductService();
```

```
public abstract getProduct();
```

```
//listener
```

```
public onCreateProduct(IRequestCycle cycle){  
    getProductService().add(getProduct());  
}
```

```
}
```



Books

Apache Wicket (<http://wicket.apache.org>)

Pro Wicket, Karthik Gurumurthy

Wicket in Action , Eelco Hillenius & Martijn Dashorst

Enjoying Web Development with Wicket, Kent Tong

Apache Tapestry (<http://tapestry.apache.org>)

Tapestry In Action, Howard Lewis Ship

Tapestry 101, Warne Onsite

Enjoying Web Development with Tapestry, Kent Tong

JSF (<http://java.sun.com/javaserverfaces>)

Core Java Server Faces, 2nd Edition, David Geary & Cay Horstmann

JSF, The Complete Reference, Chris Schalk & Ed Burns



Avoid FUD and illogical reasoning..

Q) “Why is JSF better than Wicket (and by that logic everything else in the world) ? ”

A) “Well, uhhm because JSF is a standard”

http://www.theserverside.com/news/thread.tss?thread_id=45345

“I can stand brute force, but brute reason is quite unbearable”

- Oscar Wilde



Try it out...

- Its always good to know about other (sometimes better?) options that are out there
- Get the distributions and Start coding and figure for yourself!

