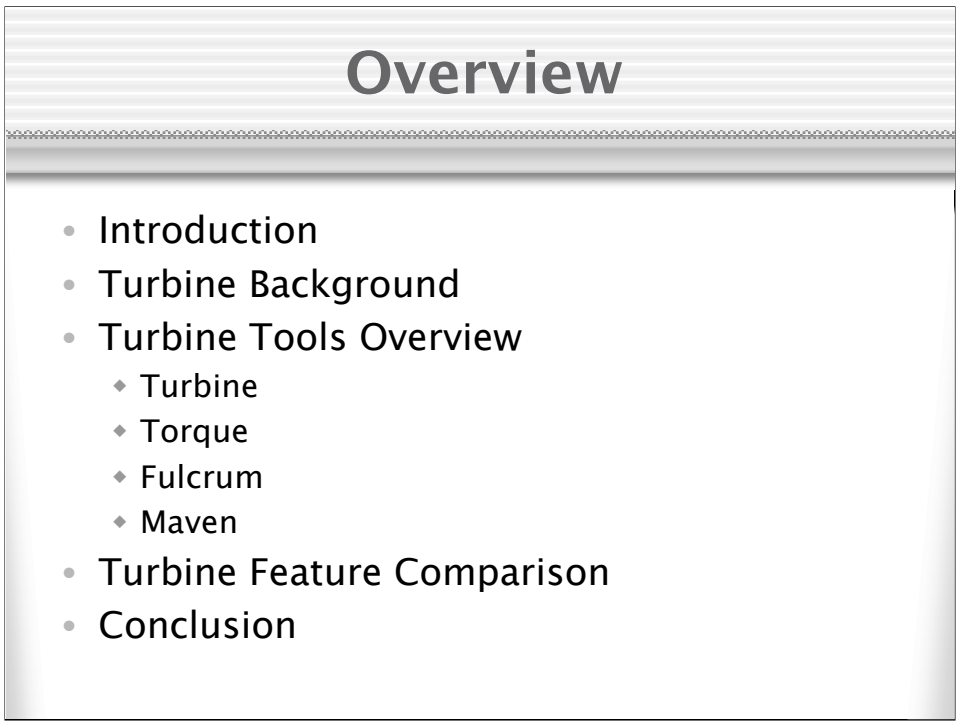
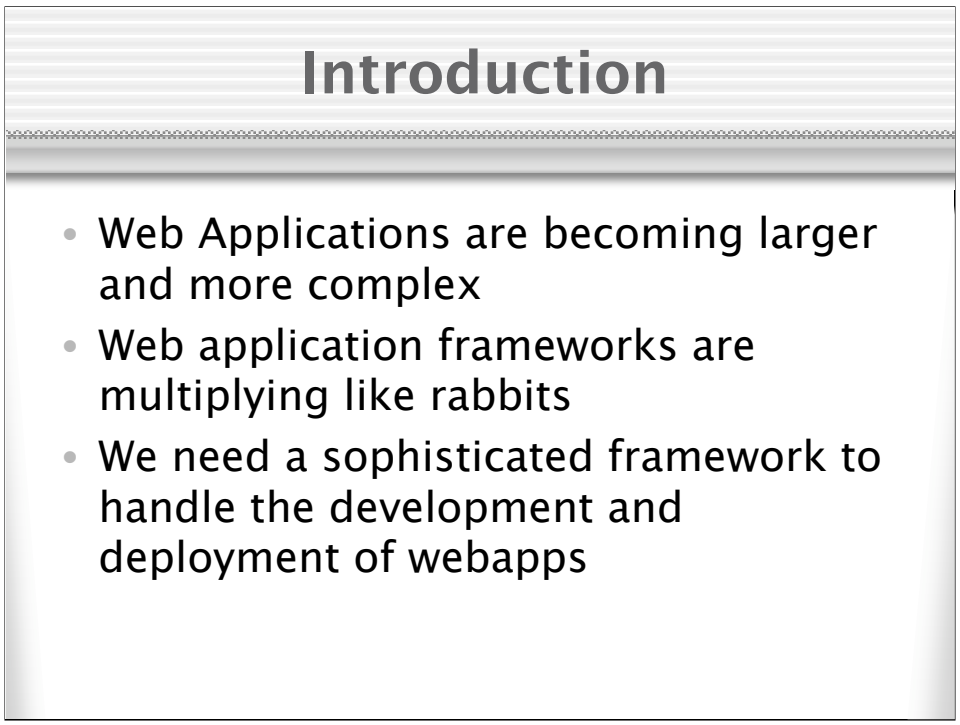
Building Web Applications With the Turbine Suite of Tools

Kurt Schrader
ApacheCon 2002
Las Vegas, NV
November 21, 2002

- Kurt Schrader
- Currently a programmer working for the University of Michigan.
- Been involved in the Turbine community as a user and a developer for a little over two years.



Starting out, I'll give a brief overview of writing webapps and touch on some of the problems people have had while writing them. Next I'll go into the background of Turbine and how it originally came about. Then I'll be discussing the 4 main Turbine tools and how they relate to one another. Finally I'll wrap things up with a brief comparison with other frameworks, and a conclusion.



Web Applications are becoming larger and more complex

o We need a solution for building webapps in a fast and easily manageable fashion.

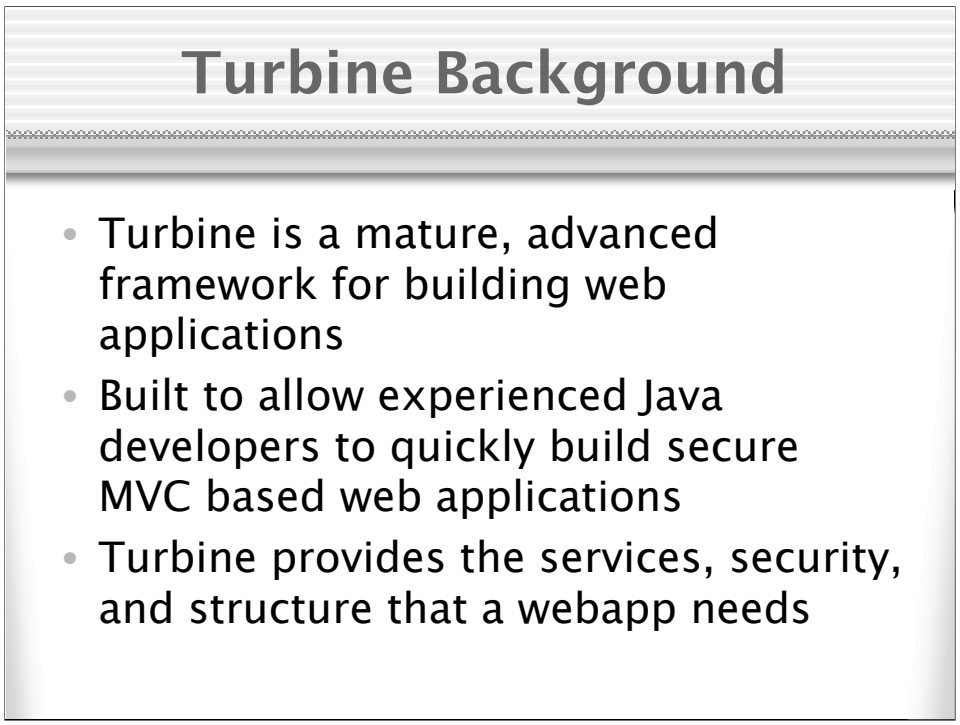
o Webapps need to be designed to not only be easy to develop, but also to maintain and expand.

Web application frameworks are multiplying like rabbits

o Well, they were multiplying like rabbits, the market seems to have settled of late.

o We currently have Struts, Cocoon, Barracuda/Enhydra, and any number of others.

What is needed is a sophisticated framework to handle the development and deployment of webapps. It needs to allow us to get started quickly, maintain and expand our application easily, and adapt to changes in the specification and requirements without tearing apart what we've already done.



Turbine is a mature, advanced framework for building web applications

o It is used for many real applications.

o It has been in development for 4 or 5 years.

o The Turbine developers have built many apps, made the mistakes, and learned from them.

o Lots of developer's experiences are rolled back into Turbine.

Built to allow experienced Java developers to quickly build secure MVC based web applications

o Turbine cleanly separates the Model, View, and Controller part of a webapp.

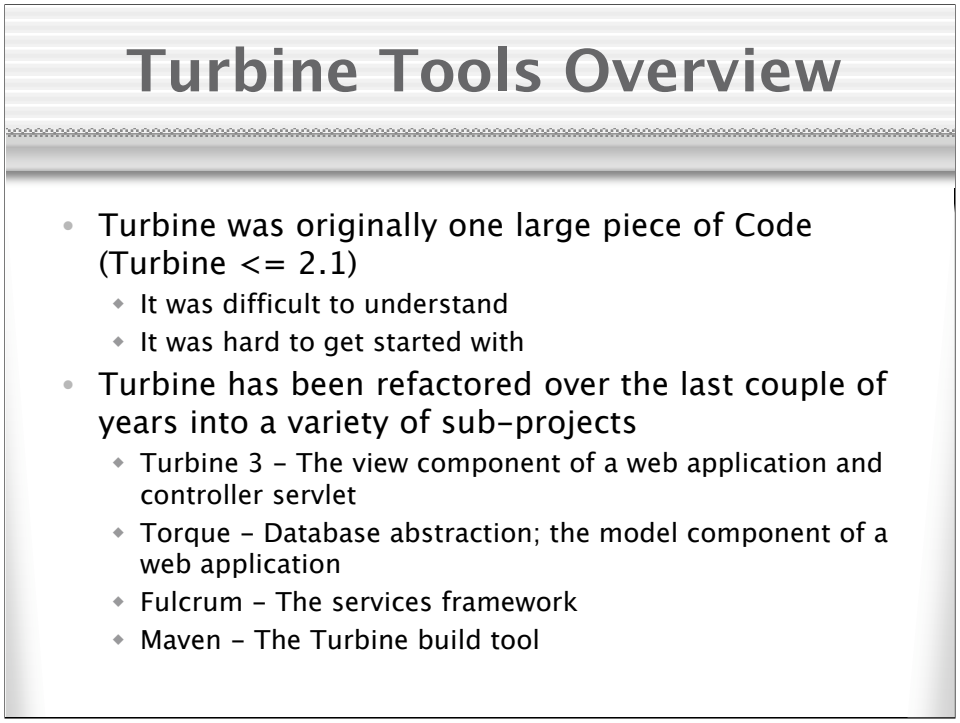
o It allows a project to get started quickly.

o It helps to manage all of the backstage pieces of building and managing a project.

Turbine provides the services, security, and structure that a webapp needs

o It provides many of the services (scheduling, user management, caching) that a webapp needs

o It allows you to define a structure for page layout at the beginning and adjust it as you go along.



Turbine was originally one large piece of Code (Turbine <= 2.1)

o It had over 300 classes before being broken apart.

o The learning curve for Turbine 2 was fairly steep

o Made it hard to see the trees from the forest. Many features were simply overlooked because there were so many of them.

Turbine has been refactored over the last couple of years into a variety of sub-projects

o Turbine 3 - The view component of a web application and the controller servlet.

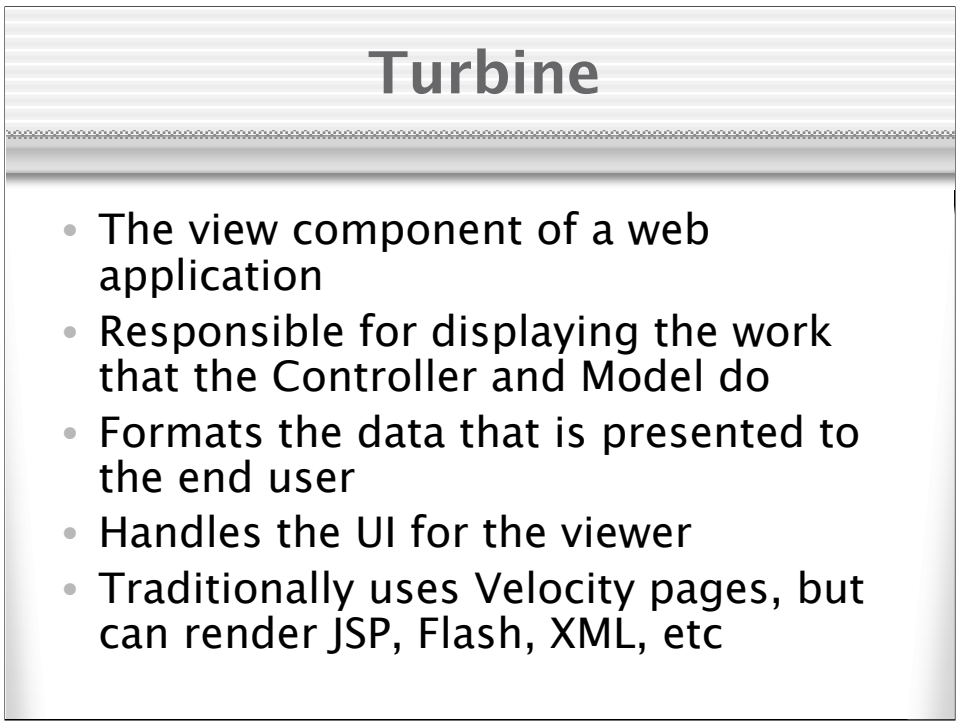
o Torque - Database abstraction used as the model component of a web application

o Fulcrum - The services framework, provides a number of services that an app would need.

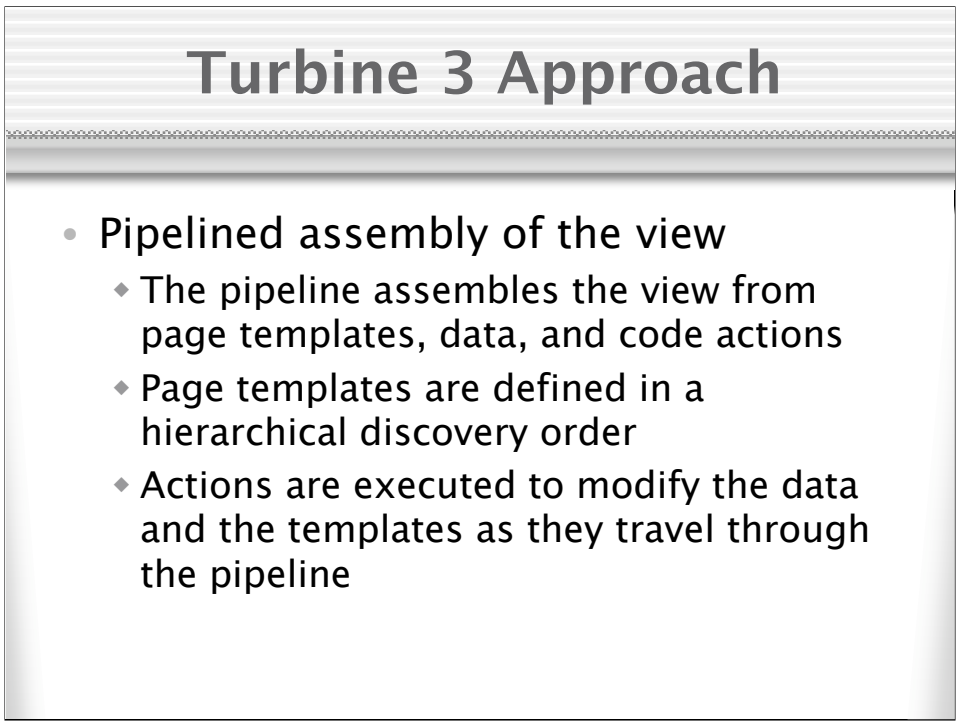
o Maven - The Turbine build tool, used for project management and comprehension.

o Any of the components can now be used independently of the others.

o Some of them seem to be growing bigger than Turbine itself.



Turbine should be seen as the view component of a web application. Traditionally the Turbine community has used the Velocity templating language to display pages, but any language that can eventually be viewed by a web browser will do. Turbine has been known to render JSP, Flash, XML, Excel, as well as other document formats. Turbine makes the data that is presented to the end user look the way it is supposed to, by formatting it in the context of the web app and producing the UI around it.

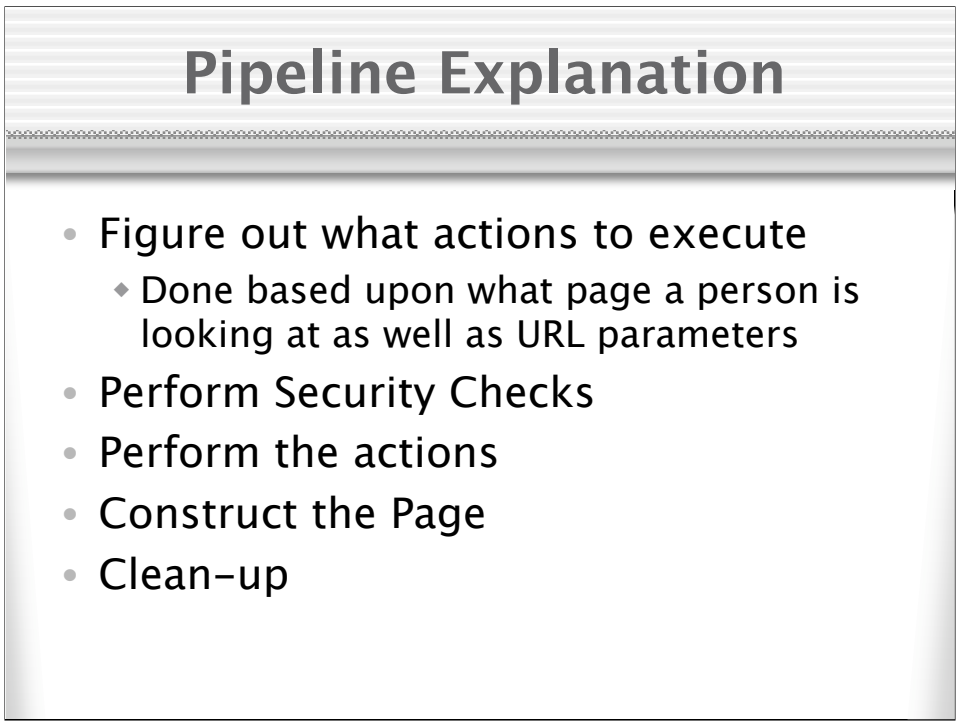


Turbine 3 traditionally assembles your view from a variety of Templates within the webapp. Each template is a fragment of a webpage or a document. These fragments are assembled together into a document as they progress through the pipeline defined by the application. The pipeline takes the proper document fragment, runs whatever java methods are needed to insert dynamic data into it, and then inserts the fragment into the overall document being formed.

Example Pipeline

```
<pipeline>
  <name>DemoPipeline</name>
  <valves>
    <valve className="org.apache.turbine.pipeline.DetermineActionValve"/>
    <valve className="org.apache.turbine.pipeline.DetermineTargetValve"/>
    <valve className="org.apache.turbine.pipeline.DefaultSessionTimeoutValve"/>
    <valve className="org.apache.turbine.pipeline.DefaultLoginValve"/>
    <valve className="org.apache.turbine.pipeline.DefaultSessionValidationValve"/>
    <valve className="org.apache.turbine.pipeline.DefaultACLCreationValve"/>
    <valve className="org.apache.turbine.pipeline.DefaultActionValve"/>
    <valve className="org.apache.turbine.pipeline.RunModulesValve"/>
    <valve className="org.apache.turbine.pipeline.DefaultTargetValve"/>
    <valve className="org.apache.turbine.pipeline.CleanUpValve"/>
  </valves>
</pipeline>
```

This is an example pipeline that is similar to the default Turbine pipeline. It is 100% customizable, however, based upon the needs of your application. Some developers have replaced all of the default Turbine valves with their own custom valves as they went along building their apps. The execution of this pipeline is explained on the next page.

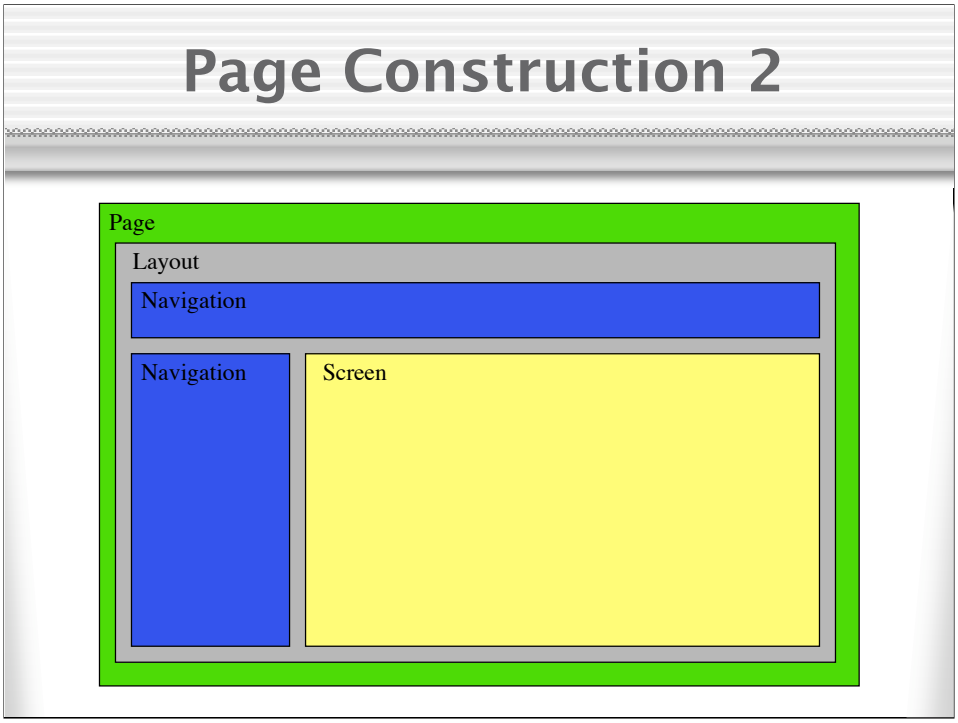


The example pipeline shown above starts out by deciding what actions (code) to execute. The action to execute is determined based upon directions given in the URL, as well as the page that the user is currently on. The default resolving algorithm can, of course, be replaced by another one if you need to use something else in your application. The pipeline then proceeds through a variety of security checks to make sure that the user is authorized to access the part of the app that they're trying to get to. Finally, we run the code, build the actual page to be displayed, and clean up.

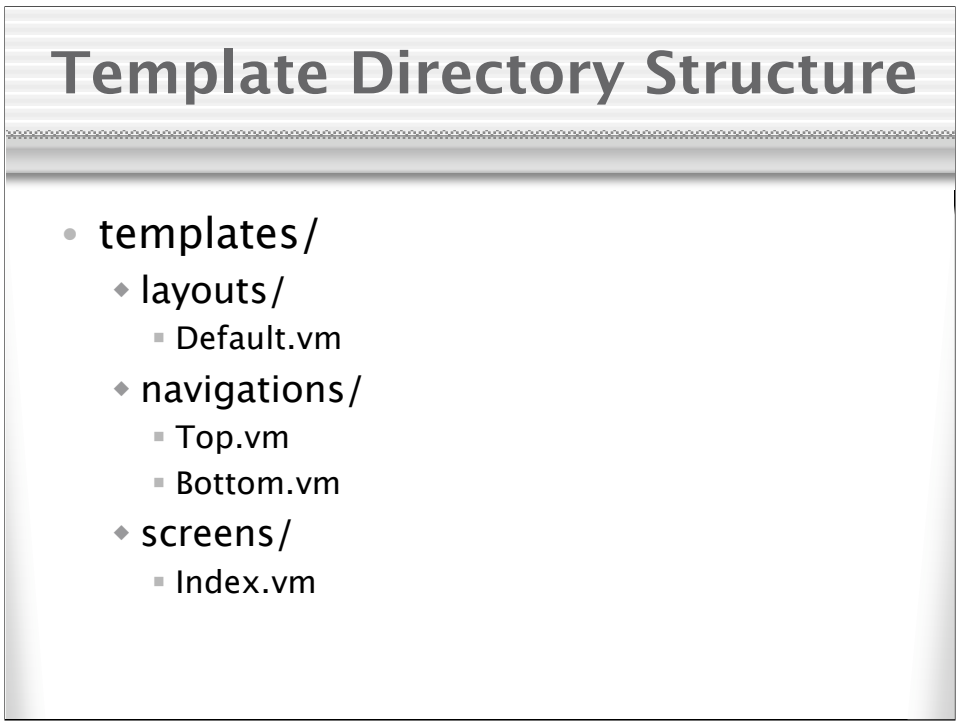
Page Construction

- Page assembly in Turbine is traditionally done using a hierarchical structure
- The pages are built from predefined templates
- The page encapsulates a layout, which in turn encapsulates navigations and screens

The traditional way to assemble a page in Turbine is from a collection of hierarchically defined folders that contain templates that represent small portions of a webpage. These templates are pieced together in a predefined manner, which traditionally has been laying out a design for the page, followed by the insertion of navigations and a screen into that layout.



Here is a picture of the traditional way that a page is assembled in Turbine. The layout is filled with navigations and a screen to give the webapp a unified and consistent appearance.

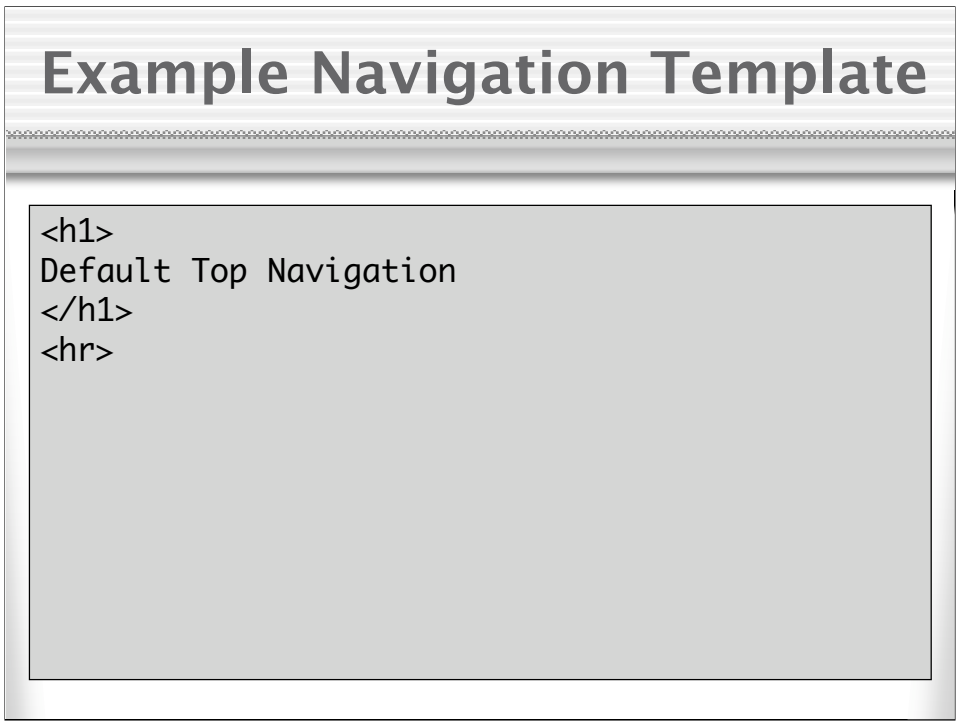


This is the way that the template directory structure has traditionally been laid out. This is an artifact from Turbine 2.X and earlier, which essentially had a fixed pipeline that only rendered things from this layout. With the advent of the separate pipeline in Turbine 3 the layout of the directories can be arranged in any way that the developer wants. This layout seems to work well for most webapps though.

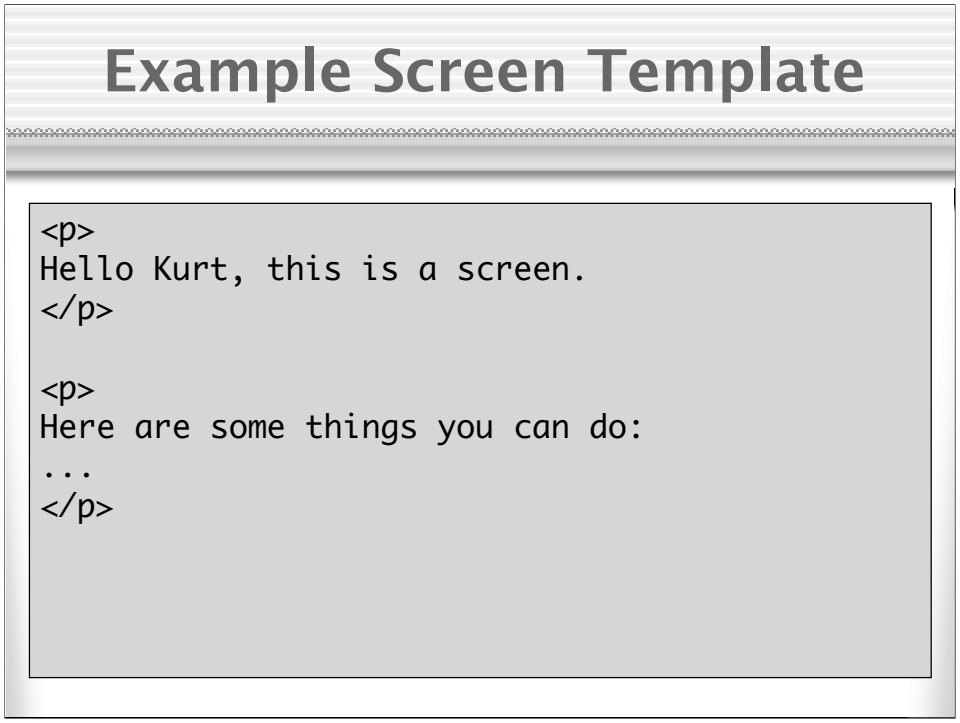
Example Layout Template

```
<html>
<head>
<title>Example App</title>
</head>
<body>
<table  width="100%”>
  <tr>
    <td>
      $renderer.render("navigations", $template, "/DefaultTop.vm")
    </td>
  </tr>
  <tr>
    <td>
      $renderer.render("screens", $template)
    </td>
  </tr>
</table>
</body>
</html>
```

This is a fragment of HTML that is used to render the outermost layout of a webpage. The calls to $renderer will be replaced with fragments of HTML as the response goes through the pipeline.

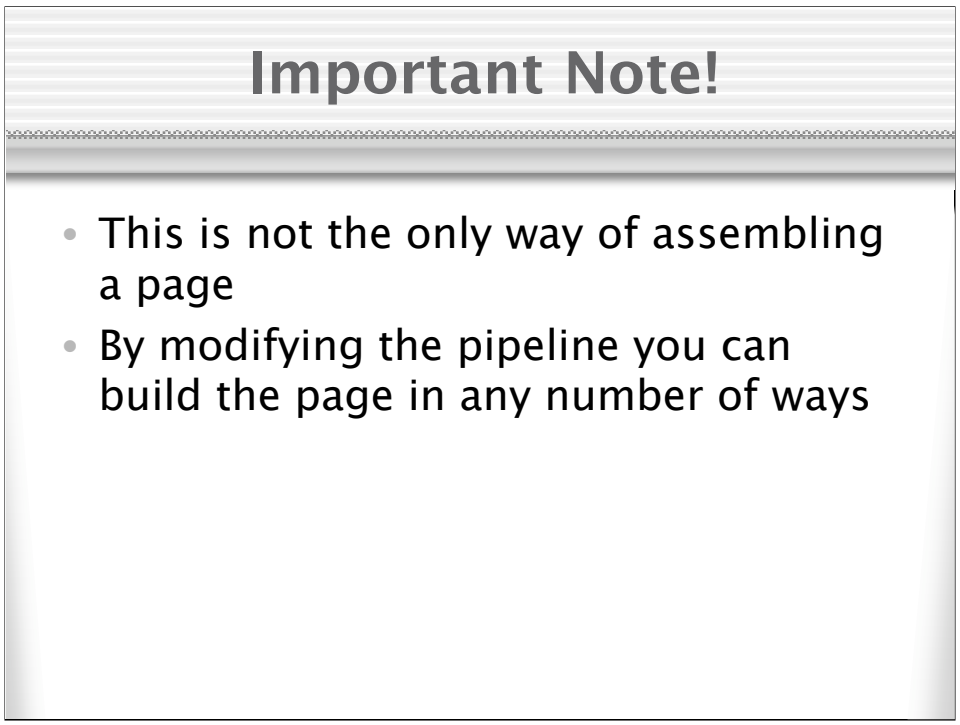


Here's a laughably simple Navigation. Usually you would have your webapp logo and company name here, or this is a side navigation then you would probably have a global menu for your application.

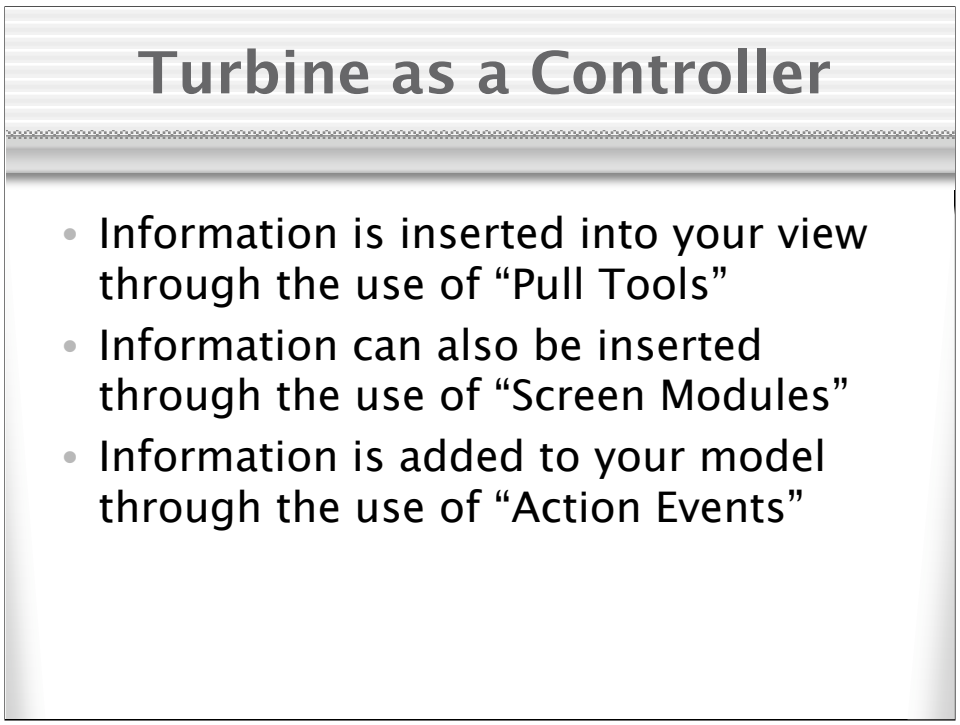


This is a simple screen. The screen is usually the focal point of your page.

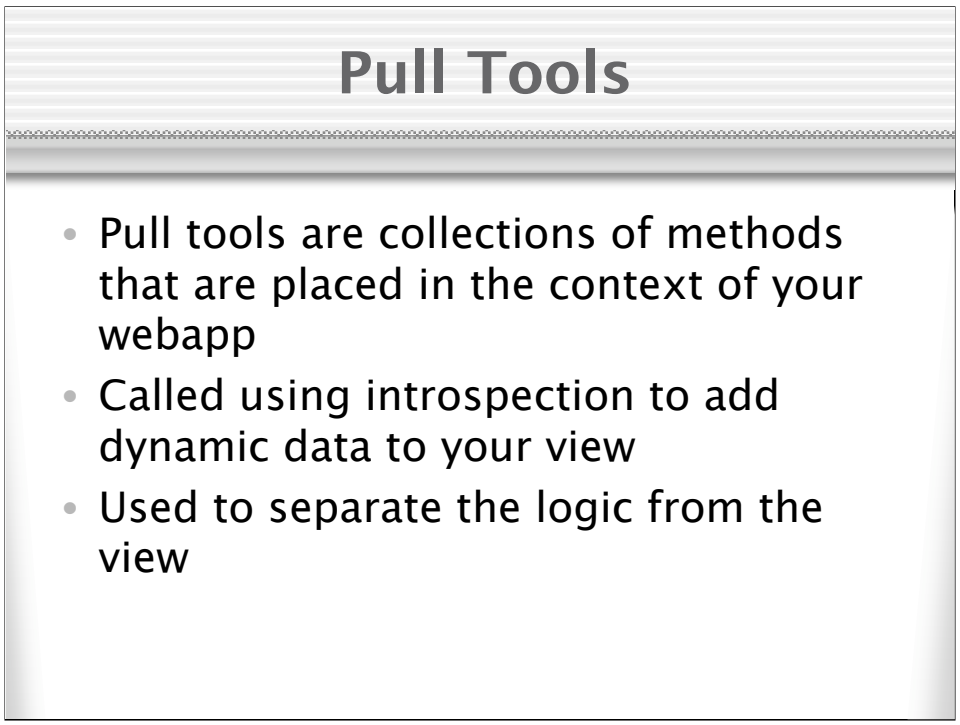
Here's what those easy examples look like when they're fed through our Pipeline and into a web browser. As you can see, it's fairly simple, but the potential is there to do whatever you want with it.



If I haven't made it clear already, this is only one of the possible ways that you can assemble your pages. You could have a pipeline that constructs a velocity page, runs it through velocity on the fly, adds in some JSP tags, and then sends it to the servlet engine for compilation and processing. I wouldn't recommend doing something like that, but the possibilities of what you can do with the pipeline are almost infinite. This is just the model that we've used traditionally in the Turbine community and it is a good example of how Turbine works. We'd like nothing more than for some people to come along and come up with some cool new pipeline configurations for Turbine.



Turbine also acts as the controller for your webapps. All requests come in through it and all output flows back out through it. This is good from a security and control standpoint, as you only have to worry about one point of access. It also shuttles information from the view to the model and from the model back to the view. There are three main ways of moving the data: pull tools, screen modules, and action events.



Pull tools are collections of methods that are placed in the context of your webapp. These methods can be called by any template to return dynamic data to them. They are called using introspection and are used in order to separate the logic from the view. Velocity, by design, only allows a few simple looping and branching constructs to manipulate data. There is no easy way to embed logic into your webpages using it.

Pull Tools Example

```
public class DateTool
    implements Recyclable, ApplicationTool{

    public String getMonth()
    {
        DateUtil dateUtil = new DateUtil();
        return dateUtil.getMonth();
    }

    public int getDay()
    {
        DateUtil dateUtil = new DateUtil();
        return dateUtil.getDay();
    }
}
```

Here is a trivial example pull tool that defines two methods; getMonth and getDay.

Pull Tools Example (cont)

- The DateTool class is added to the application context as “date”
- Calling “$date.Day” returns an int representing the current day
- Calling “$date.Month” returns a String representing the month
- Pull tools are not tied to any particular screen

The DateTool class is then added to the application context by using a configuration file at start-up time. Once a tool is added to the context, any of its methods can be called by any of the templates to add dynamic data to the application.

Screen Modules

- Screen modules are tied to a specific template
- They execute code whenever that screen is viewed and make the results available to the screen
- You can also have default screen modules that execute on pages with no specific module tied to them

Screen modules, on the other hand, are tied to a specific template. They execute the code contained within them whenever that specific screen is accessed. This separates the code needed by the view from the view itself, but makes code reuse harder and less intuitive. The screen modules system also supports default modules that execute whenever a screen with no specific class tied to it are accessed.

Screen Modules Example

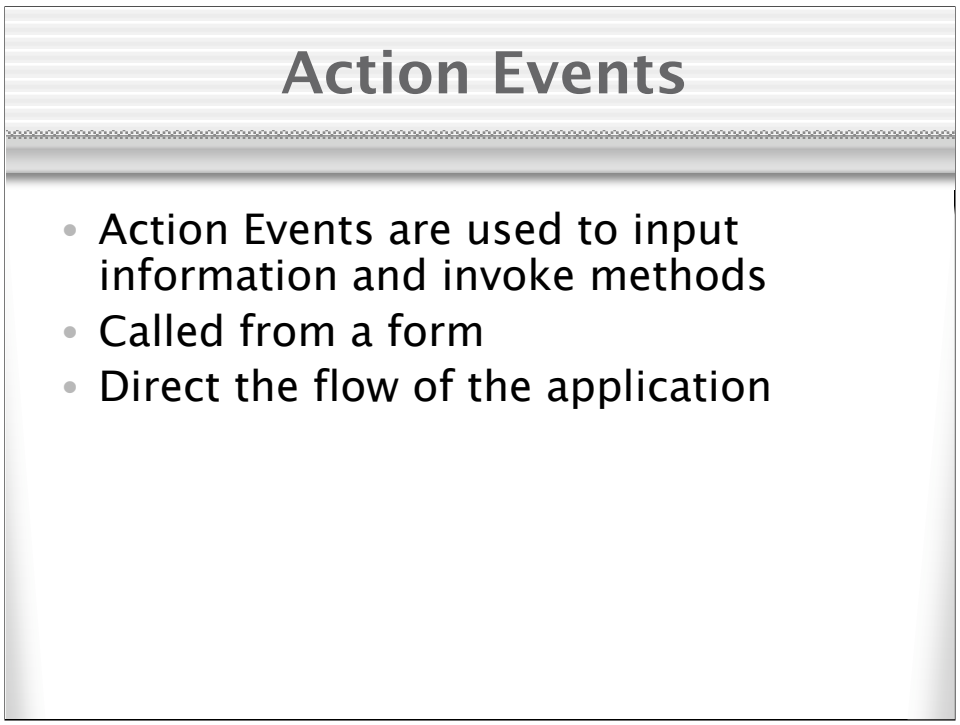
```
public class Date
    extends TemplateScreen{

    public void doBuildTemplate(RunData data, Context context)
    {
        DateUtil dateUtil = new DateUtil();
        context.put("month", dateUtil.getMonth());
        context.put("day", dateUtil.getDay());
    }
}
```

Here is a trivial example screen module that puts the current month and date into the context.

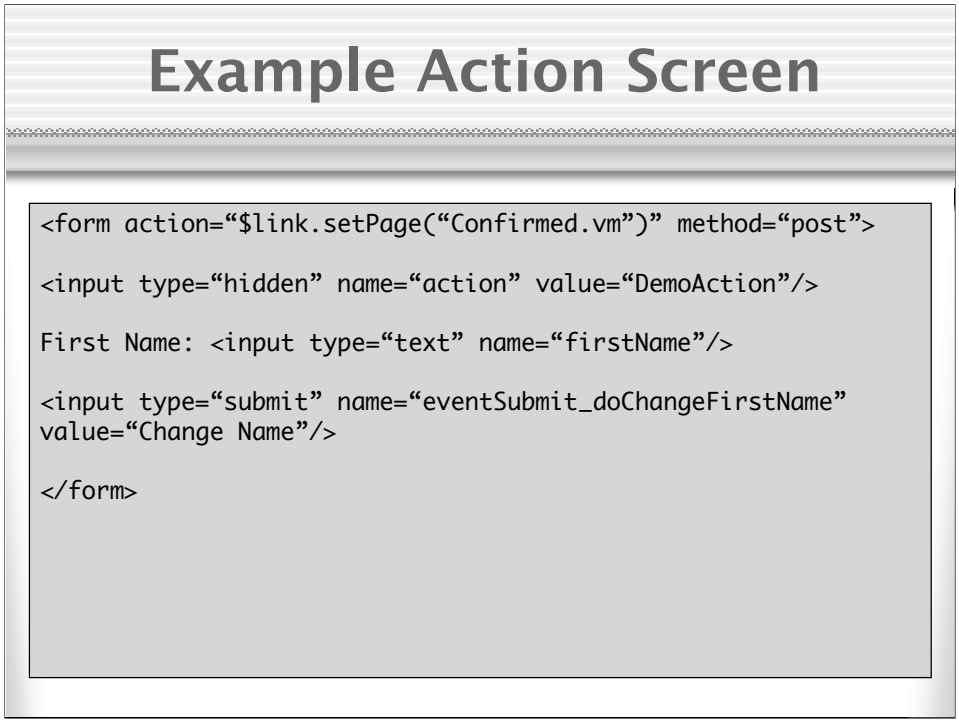
Screen Modules (Cont)

- The “Date.vm” can now access these values
- The month resolves as “$month”
- The day resolves as “$day”
- Tied to a specific screen

The Date.vm page (in the default Turbine resolving scheme) can now access the current day and month using the variables designated in the class. Unfortunately, this logic is tied to this specific screen and has to be added to the screens for all of the other pages that want to use it as well. This shortcoming is the reason that pull tools were originally developed.



Action events are called when you need to move information from a form into the application model. They pull the data in from the form and manipulate it as necessary. They can also change the flow of the application by sending someone to a different page depending the input.

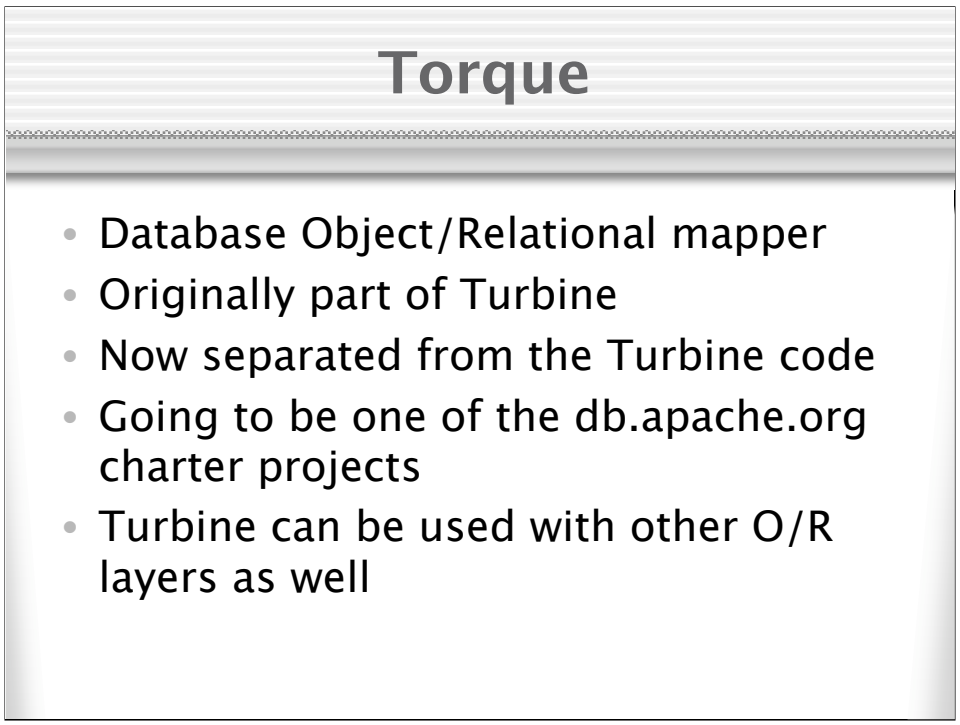


Here is an example of a simple form that is calling an action event. The action is "DemoAction" and the name of the method is "doChangeFirstName." This method is called automatically if you set up your forms in this way.

Example Action Class

```
public class DemoAction extends Action
{
    public void doChangeFirstName(RunData data, TemplateContext context)
            throws Exception
    {
        String newName = data.getParameters().getString("firstName");
        Person.changeFirstName(newName);
    }

    public void doPerform(RunData data, TemplateContext context)
            throws Exception
    {
        log.error(”DemoAction called an empty method!");
    }
}
```

Here’s the action class that was called by the last form. As you can see, it’s simply getting the parameter that passed in and sending it to the proper method. The “doPerfom” method is called if an action is run but no method matches the name that you’re specified in your action event. It’s usually seen as an error whenever the doPerform method is called.

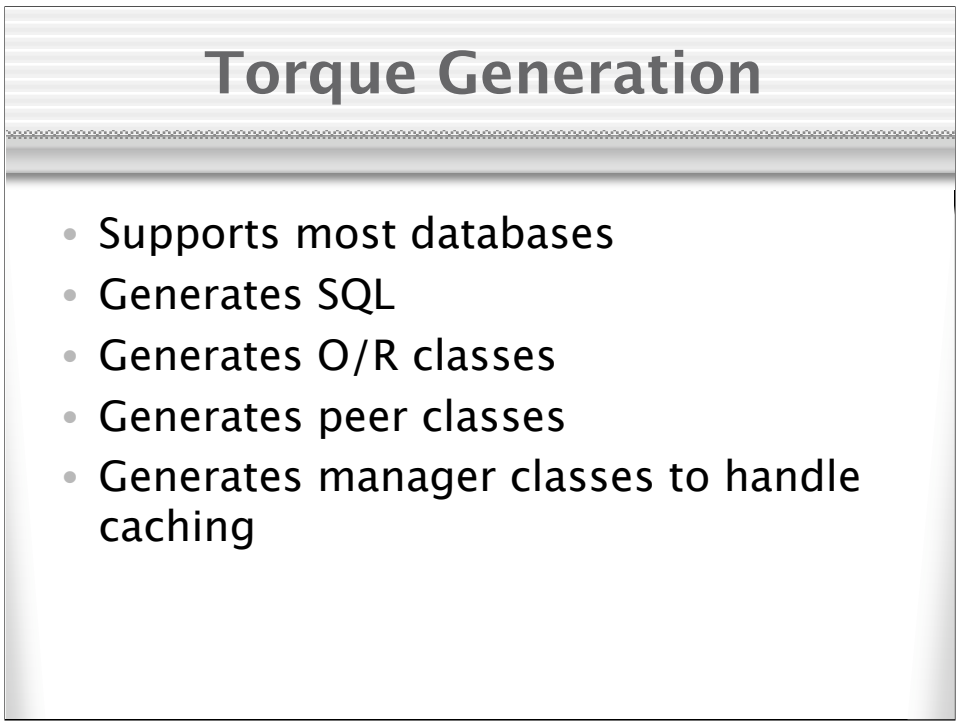


The next tool in the Turbine family, Torque, is the Turbine object relation mapping framework. It was originally part of Turbine but was separated from the main code base when we decided to split it. It has now grown into a large project in its own right and is going to be one of the charter projects of the db.apache.org project. Although Torque is the de facto O/R layer for use with Turbine, any O/R layer can easily be used.

Example Torque Schema

```
<database name=”example">
  <table name=”PERSON" idMethod=”native">
    <column name="UNIQID" required="true"
        primaryKey="true" type="INTEGER"/>
    <column name=”FIRST_NAME" type=”STRING"/>
    <column name=”LAST_NAME" type=”STRING"/>
    <column name=”EMAIL" type=”STRING"/>
    <column name=”LAST_LOGIN" type=”DATE"/>
  </table>
<database>
```

This is an example torque schema. It describes the layout of only one table. In a real torque schema, you would specify many tables and the relationships between them. This schema can also be reverse engineered from an existing database using a built in torque tool for doing so.

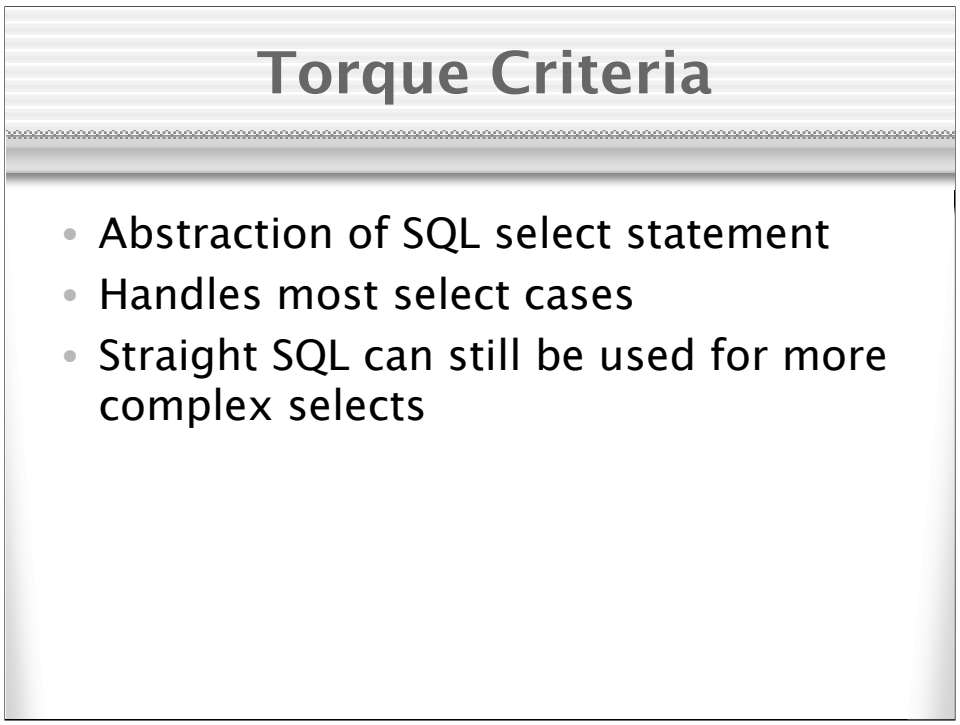


Using the Torque schema, you can generate the SQL and classes to map your database to objects. The generation process also produces Peer classes that have many common functions in them (doSelect, doDelete, etc), as well as manager classes that handle the caching of objects in memory.

Using Generated Classes

```
Person person = new Person();

person.setFirstName("Kurt");

person.setLastName("Schrader");

person.save();
```

This is a simple example of using one of the generated classes to insert a person into the database using the generated Torque objects.

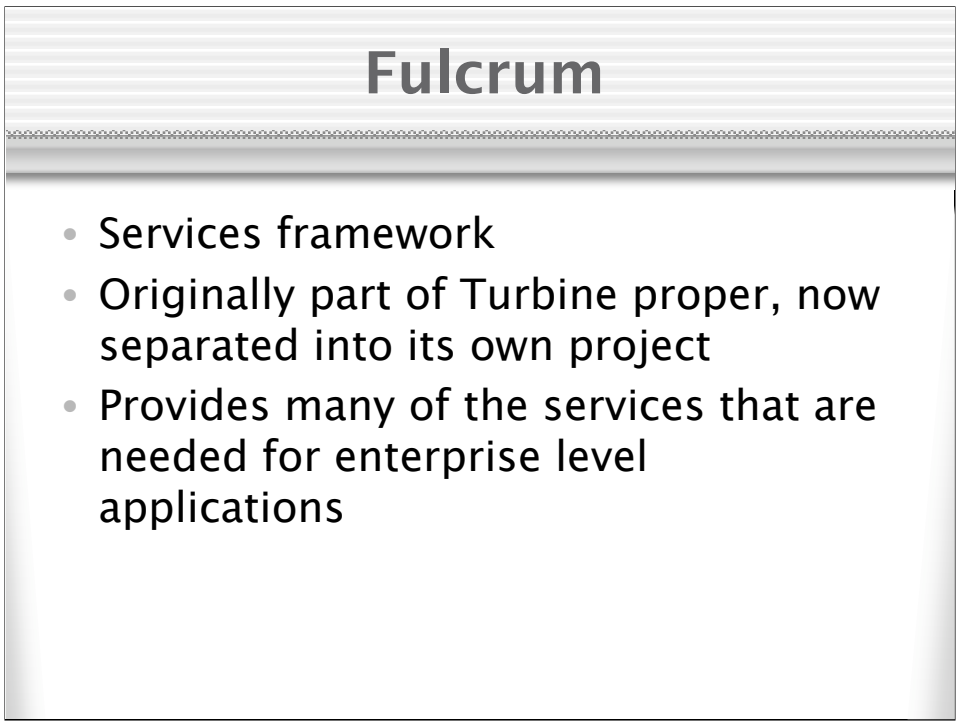


The Torque Criteria object is an abstraction of a SQL select statement. It can used to select objects out of the database for most common queries. If a query is needed that is more complex than those that can be constructed with the criteria object, straight SQL can still be used.

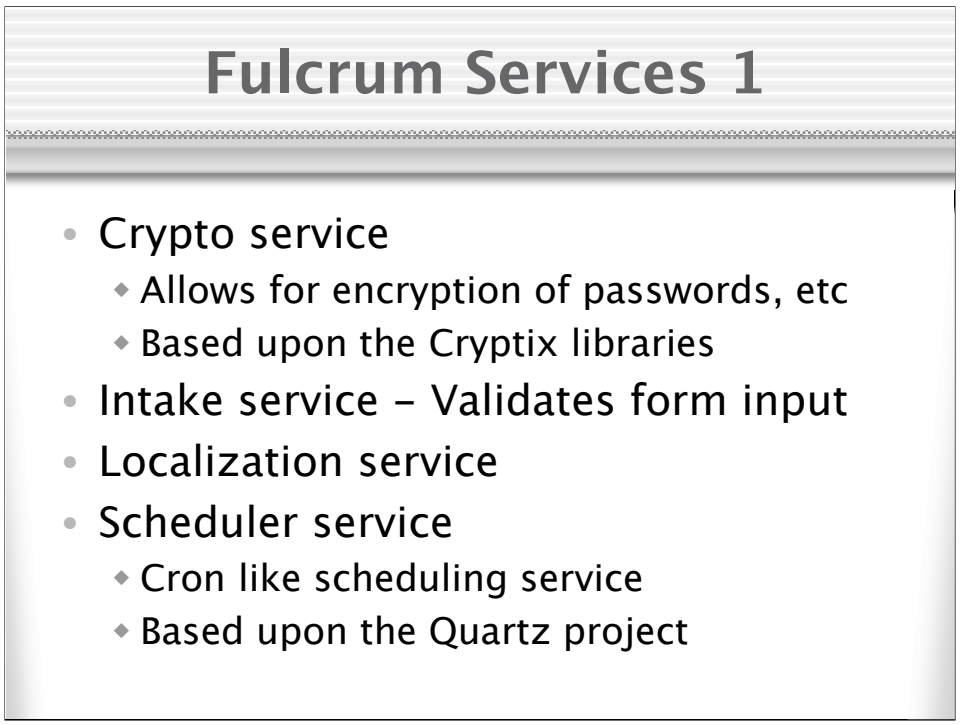
Criteria Example

```
Criteria criteria = new Criteria();

criteria.add(PersonPeer.FIRST_NAME, “Kurt”);

criteria.add(PersonPeer.LAST_NAME, “Schrader”);

List people = PersonPeer.doSelect(criteria);
```

This is an example of using the Criteria object. It will select all of the people in the database with the first name “Kurt” and the last name “Schrader”.



Shifting gears once again, we come to Fulcrum, the Turbine services framework. Fulcrum brings to the table many of the services that one needs to build an enterprise application. Fulcrum was also originally part of the Turbine code base, but now the code has been split into its own project that can be used with any application.



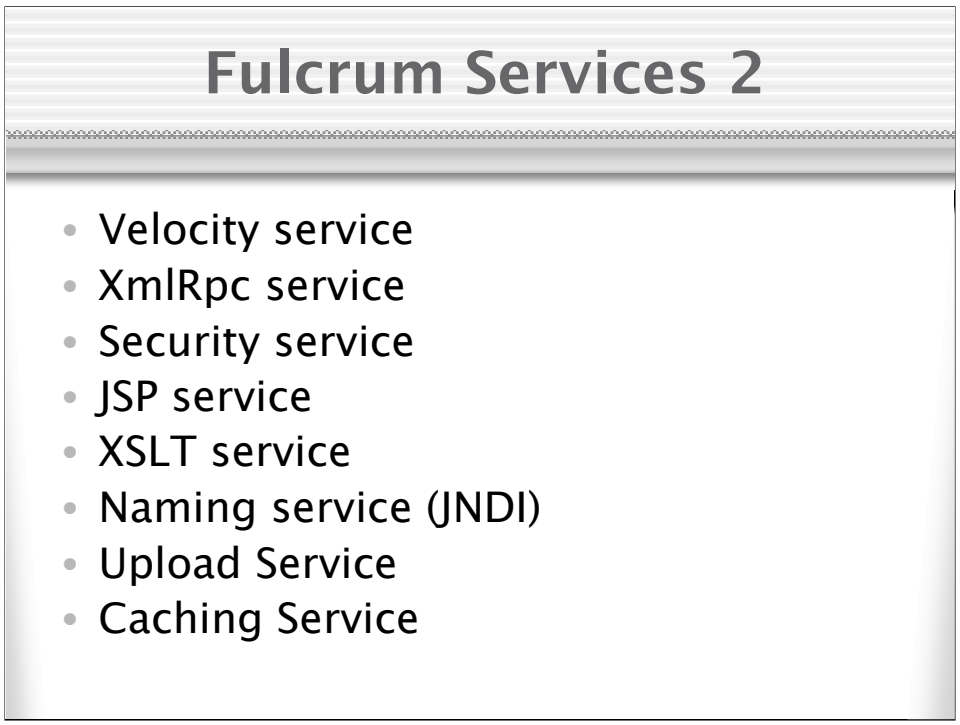
Some of the services that fulcrum provides include:

o A crypto service for encrypting incoming data before storing it.

o The “Intake” service which is used as a framework for validation input that users enter into forms.

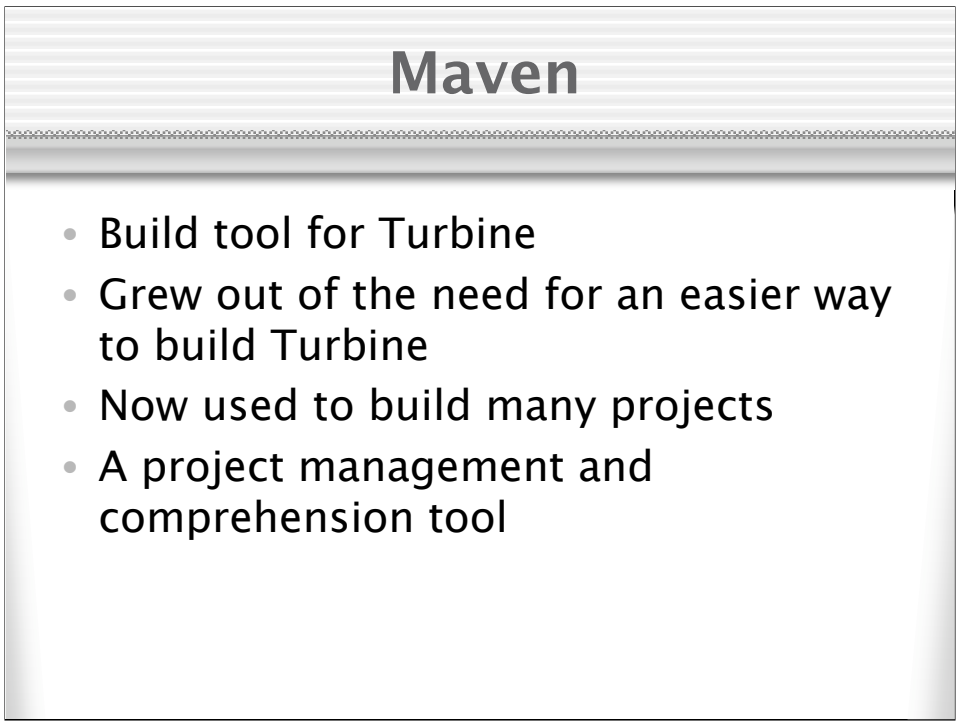
o A localization service to convert your text into other languages.

o A scheduler service to schedule jobs that need to run at a given time.

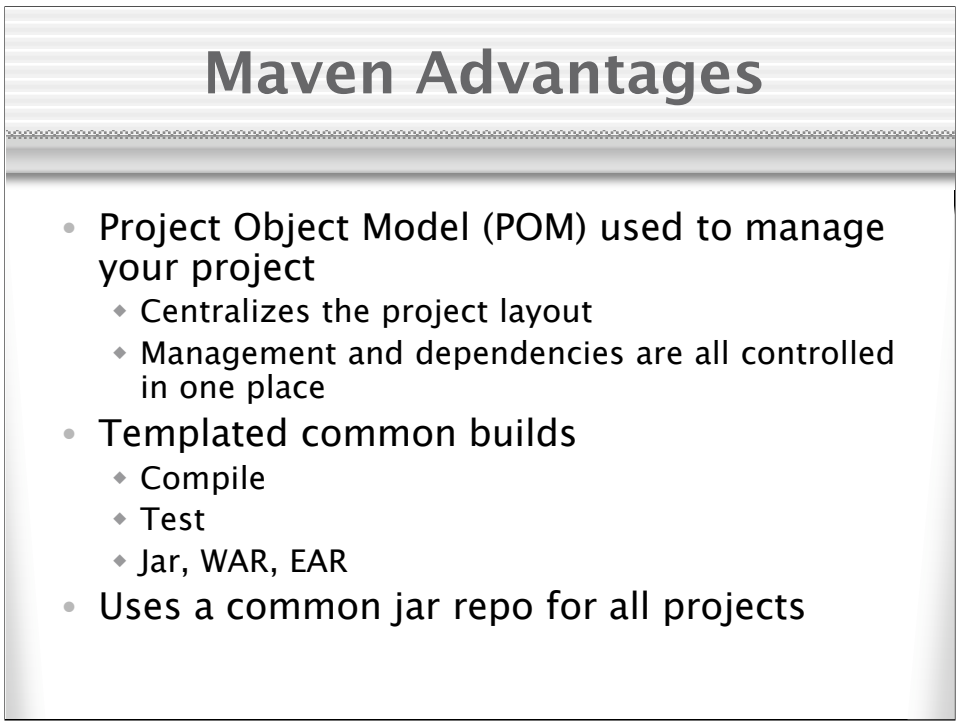


Some other Fulcrum services are:

- Velocity service: Render Velocity templates
- XmlRpc service: Make XmlRpc calls to another server.
- Security service: Has the concept of Users, Groups, and Roles to authenticate against.
- JSP service: Render JSP pages
- XSLT service: Transform XML documents into another form
- Naming service (JNDI): Allows for the look-up of JNDI realms
- Upload Service: Allows for the easy upload of files from a form.
- Caching Service: Caches objects internally.



Maven was originally designed as the build tool for Turbine. It grew out of the need to easily manage and build the Turbine projects, all of which have many varying dependencies. It too, has grown into a large project and is now used to build many projects. Maven now strives to be a tool that allows for easy management of your projects as well as allowing people to quickly and easily comprehend a new project that they're getting involved in.



Maven uses a Project Object Model to manage a project under its control. The project object model is an XML document that centralizes all of the meta-information about a project. Once this information is supplied, Maven uses it to perform any number of common templated builds. These builds currently include such things as compiling your project, running the test cases, building a jar, building a war, and automatically building documentation. Maven also uses a centralized local jar repo. The advantage of a centralized local repo is that you don't have to have many different versions of the same jar spread all over your various projects.

POM Example

```
<project>
  <pomVersion>3</pomVersion>
  <id>maven</id>
  <name>Maven</name>
  <currentVersion>1.0-beta-8</currentVersion>
  <organization>
    <name>Apache Software Foundation</name>
    <url>http://jakarta.apache.org/</url>
    <logo>/images/jakarta-logo-blue.gif</logo>
  </organization>
  <inceptionYear>2001</inceptionYear>
  <package>org.apache.maven</package>
  <logo>/images/maven.jpg</logo>
```

This is the top of a Maven POM file. This is very general information about the project and the organization that it comes from.

POM Example (cont)

```
<repository>
    <connection>
  scm:cvs:pserver:anoncvs@cvs.apache.org:/home/cvspublic:jakarta-turbine-maven
  </connection>
   <url>http://cvs.apache.org/viewcvs/jakarta-turbine-maven/</url>
 </repository>

 <versions>
   <version>
     <id>b1</id>
     <name>1.0-b1</name>
     <tag>MAVEN_1_0_B1</tag>
   </version>
 </versions>
```

Next in the POM is the repository section (if you have one) and the versions section (if you have any). Both of these sections are optional.

POM Example (cont)

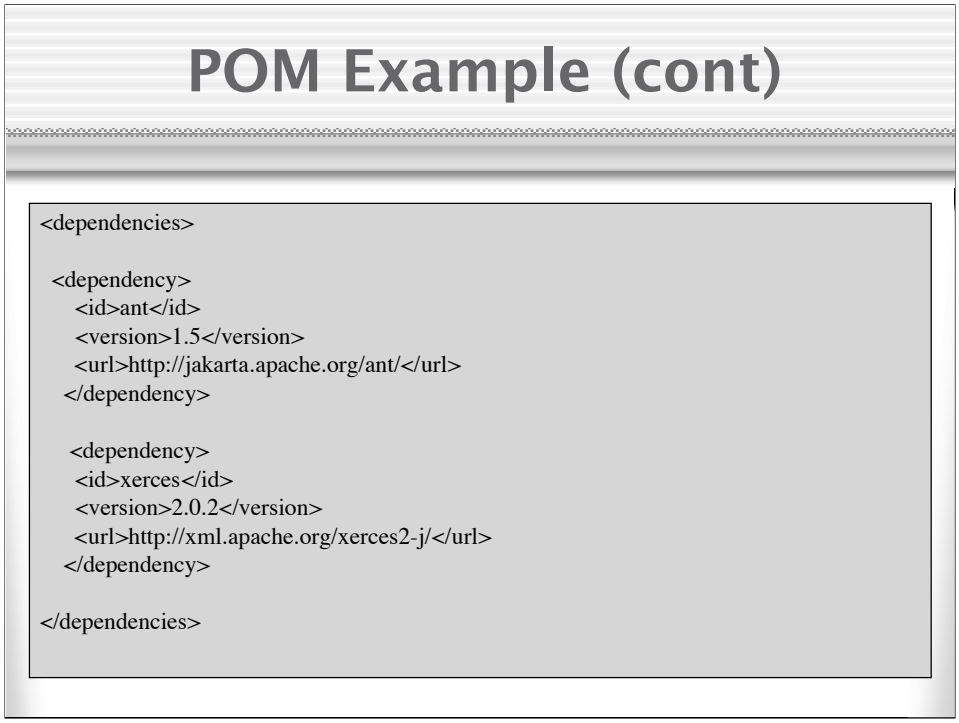
```
<mailingLists>
  <mailingList>
    <name>Maven User List</name>
    <subscribe>turbine-maven-user-subscribe@jakarta.apache.org</subscribe>
    <unsubscribe>turbine-maven-user-unsubscribe@jakarta.apache.org</unsubscribe>
    <archive>
    http://nagoya.apache.org/eyebrowse/
      SummarizeList?listName=turbine-maven-user@jakarta.apache.org
   </archive>
  </mailingList>
 </mailingLists>
```

Next you need to specify the mailing lists for the project. This step is also optional.

POM Example (cont)

```
<developers>
  <developer>
     <name>Kurt Schrader</name>
     <id>kschrader</id>
     <email>kschrader@karmalab.org</email>
     <organization>University of Michigan</organization>
     <roles>
       <role>Java Developer</role>
     </roles>
   </developer>
</developers>
```

Then you need to specify the developers on the project.

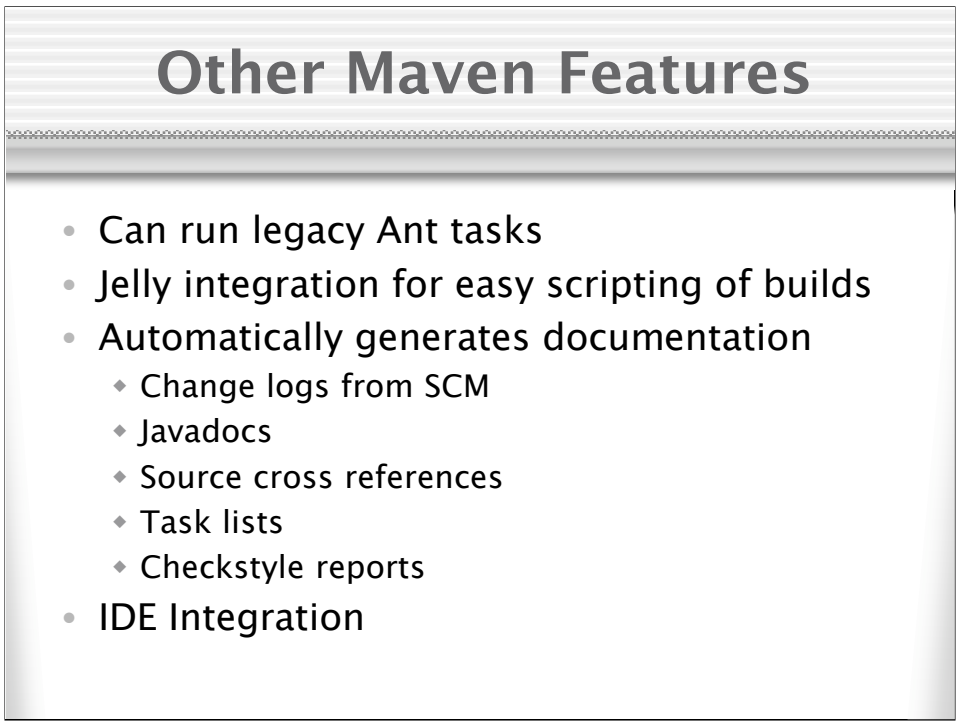


The dependencies portion of the POM is one of the most important. The dependencies noted here are used while building and testing your project. They are stored in the centralized local Maven repository and if one or more of them is not present at build time then an attempt will be made to download them from the master Maven repository on Ibiblio. You can also set up your own internal repositories to download dependencies from as well.

POM Example (cont)

```
<build>
  <sourceDirectory>src/java</sourceDirectory>
  <unitTestSourceDirectory>src/test/java</unitTestSourceDirectory>
  <unitTest>
    <includes>
      <include>**/*Test.java</include>
    </includes>
    <excludes/>
  </unitTest>
  <!-- Resources that are packaged up inside the JAR file -->
  <resources>
    <includes>
      <include>log4j.properties</include>
    </includes>
  </resources>
 </build>
</project>
```

Finally, you need to specify where your source and tests live. Maven uses this information in its build and test process to automate those tasks.

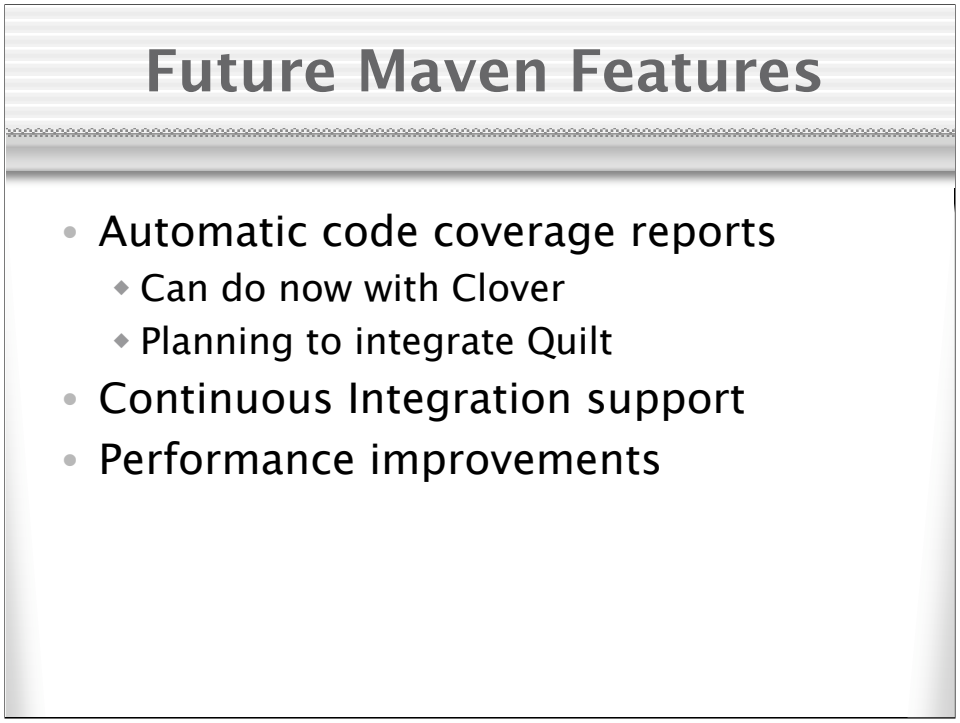


Creating a POM like the one that we just went through is all that you have to do to make use of Maven. Other features of Maven include:

The ability to run your legacy Ant tasks if you're moving over from Ant.

Automatically generating a set of standard documentation for your project.

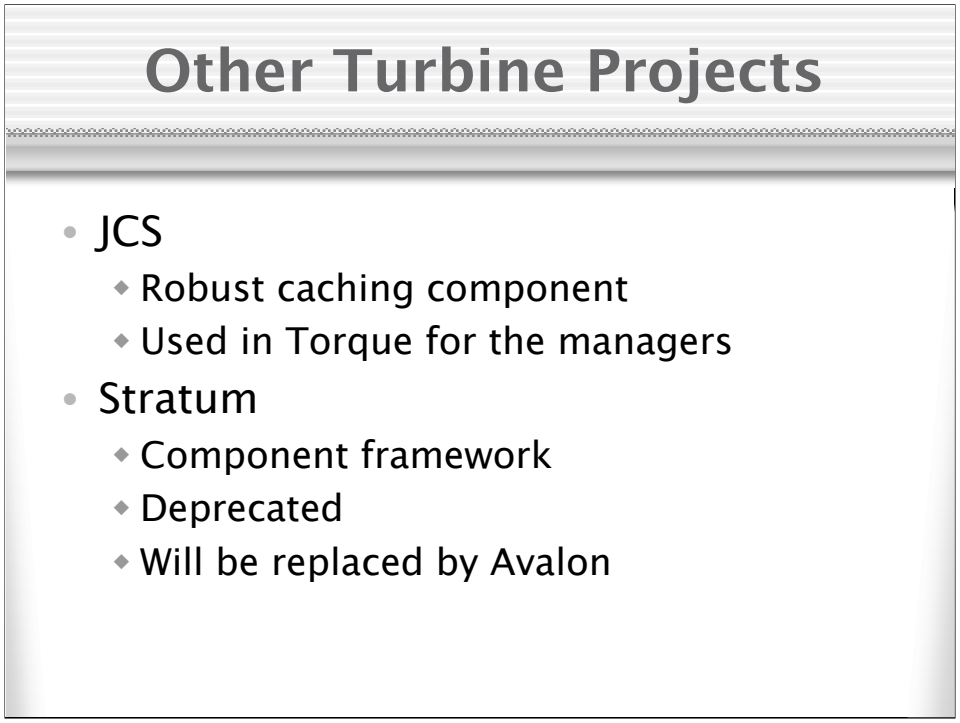
Integration with IDEA, JBuilder, and Eclipse. A project with a POM can generate the project files necessary for working with these IDEs by simply typing 'maven <ide name>' at the command line.



Maven is a fairly new project that isn't complete yet. In the future we plan to add automated code coverage reports and continuous integration support, as well as a number of other features. We also intend to work on improving the performance of Maven, as it takes some amount of time to start up currently. This start up time can avoided by using Maven in its interactive console mode to build your projects.

Example Maven Website

This is a screenshot of the website that is generated automatically by Maven. This can be done as soon as you write a POM for your project, and it can be customized through a set of properties passed into the build. The default maven documentation build creates much of the essential documentation (JavaDocs, test reports, CheckStyle reports, etc) for your project automatically.



The Turbine family also includes a couple of other projects:

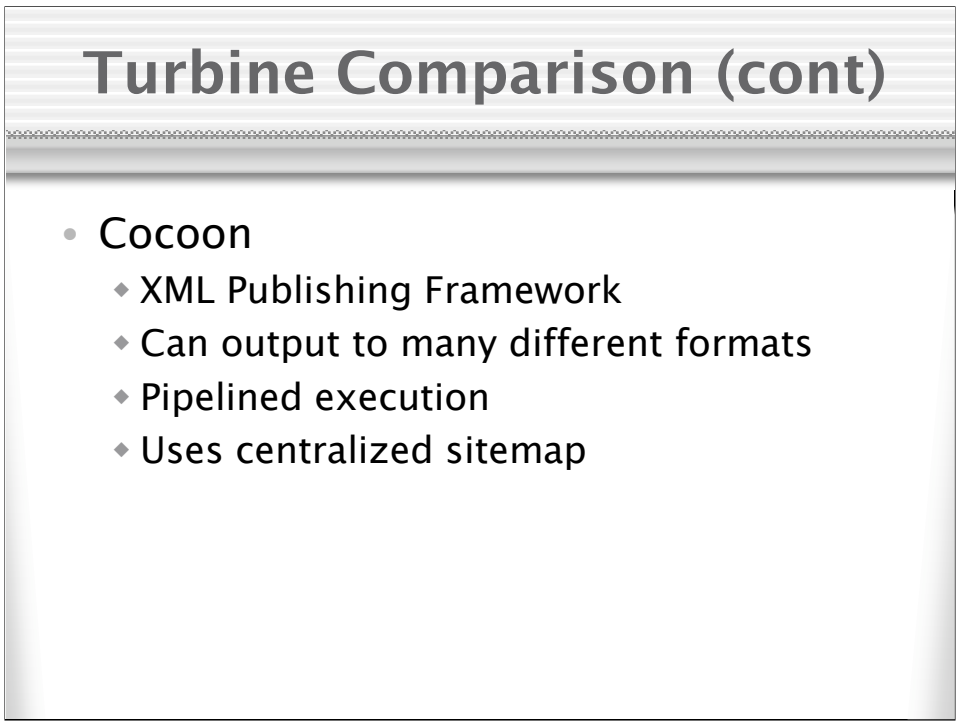
JCS is a robust caching component that provides many different types of caching functionality. It is currently used in Torque for its caching managers.

Stratum is a component framework that came about when Turbine was separated. It is currently in use but has been deprecated as the Turbine team has decided to use Avalon for managing components as we move forward.

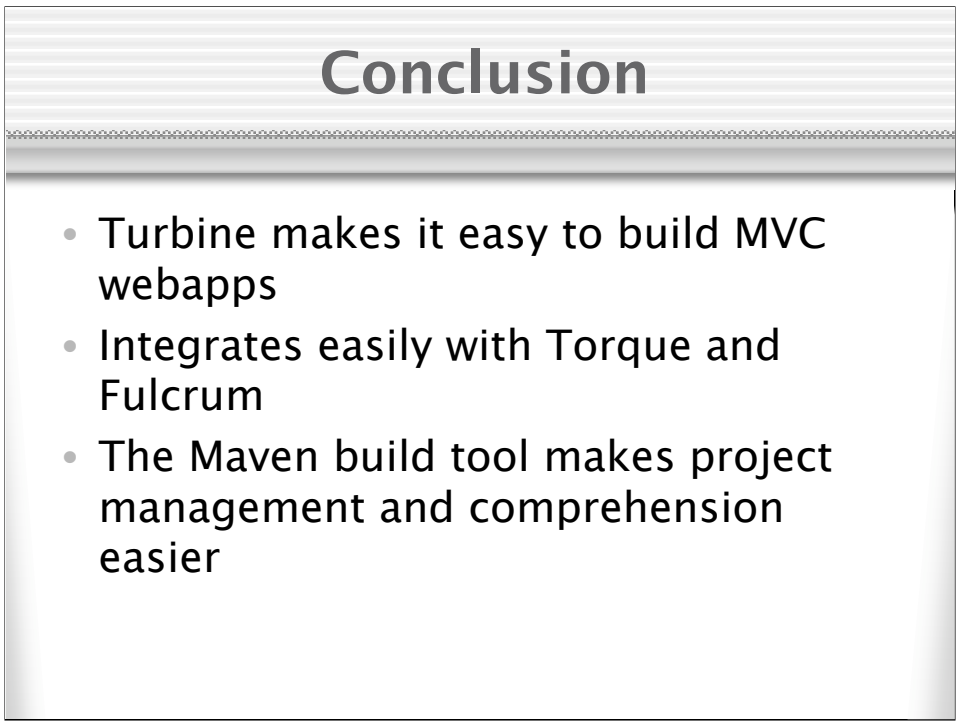
Turbine Comparison

- Struts
  - MVC Framework
  - Used most often with JSP
  - Most processing done with taglibs
  - No real idea of a pipeline for the view
  - Allows for all of the inherent problems that come from using JSP

This is a quick overview of Struts. Although I feel that Struts has a number of shortcomings when compared to Turbine, it definitely has the best support for JSPs. If you have to use JSPs in your web project then I suggest that you use Struts, as JSP support in Turbine currently leaves something to be desired.



Cocoon is a framework for converting XML data into documents. It can produce many types of output (HTML, PDF, etc) quite easily from XML. Unfortunately, it only supports XML as the input source and has a fairly steep learning curve to get started with. If your data is already in XML, however, then Cocoon is probably your best choice.



The Turbine family of projects are geared towards making the development of large web-apps easier. There is clean integration between the different Turbine projects and Maven helps to pull everything together into a coherent whole.

URLs

- http://jakarta.apache.org/turbine/
- http://jakarta.apache.org/turbine/maven/
- http://jakarta.apache.org/turbine/torque/
- http://jakarta.apache.org/turbine/fulcrum/
- http://jakarta.apache.org/velocity/