GT2004Jeremy

Cocoon Best Practices

Lessons learnt during the development of big projects Or: how to avoid stupid mistakes

Jeremy Quinn of Luminas

Developer of web applications

Combination of knowledge gathered from several sources

Usecases

Formalized way of describing how a piece of software is going to work. In the way of formalized actors (eg. User, Admin, System etc.). Luminas developed an inhouse usecase markup language. Writing use cases is the stage where everyone gets to know what needs to happen. Everyone who is involved in the project needs to flag up issues in their domain. Can be used as the base of a contract between you and the client. Keep it updated: projects change and usecases change with them. Usecases should be versioned with the software. Descriptions in usecases should be understandable by client and developer. It's an engineering document, not a marketing document. A critical point of communication between developer and client. Usecases are published on-line with the project in a private webspace

Testing

- Continuous testing
- Unit tests
- Functional tests

With unit-tests you test if the java code is working.

Sitemap

Design your URI space. URIs should be memorable. Keep the URIs simple. Design the space early in the project. It is the contract with the world. Suffixes should be used consistently, where content is used in different formats, when content is more or less static. Environment should output relative URLs. Setup the top level of the project carefully. You can't rely on what the cocoon sitemap sets up, in case of upgrading to a newer cocoon version. Top sitemap is a good place for reusable content. Using subsitemaps is important. Sitemaps should be as small as possible. Easier to maintain and debug. Resources (map:resource) is very useful.

Always use i18n

You don't want flowscript outputting text strings. Separation to concerns. People who write messages have writing skills, not development skills. Output i18n key from flowscript, jxtemplate wraps it in i18n tags. Same thing for cocoon forms. No display strings in model or template. Dictionaries, even if it is only one. Test UTF-8 very carefully. Server OS may treat them inconsistently

Relational Databases

Don't use SQL itself. Hibernate outputs very good SQL. Separates logic from the structure of the tables. Write the mapping file and use Hibernate to generate db schema and java-beans.

Flowscript

Explicitly pass parameters from the sitemap to the flowscripts. This means you make a parameter in the sitemap that gets the data, and pass each explicitly from the sitemap, so looking at the sitemap shows exactly what the contract is. Don't write business logic in flowscript. Keep the scripts short. Several pages is too long. Flow is preferred over actions, actions over XSP.

Development

Use source control management tools. Test before you commit. Commit early and often. Write proper comments. Don't check in built material or local customization. Read commit mails.

Replicate

It's important you can completely and reliable replicate a project when you check it out. Check wiki YourCocoonBasedProject . The environment that you replicate your project in should be put together in a dependable way

Bugzilla

Bugs can be seen as a rude word to a developer, don't take it personally. You can even subscribe your client to the bug e-mails. Warn your developers if you do this!

Going live

Learn how to use the logging control files. Turn of debug messages and reduce output to a minimum. Disable dangerous features. Don't run cocoon in a publicly visible way. Get apache to provide error messages and static assets. Technical errors should still be available to developers. Automate production environment with an ant task

Be a good citizen

Read the manual, search the wiki and mail list. No one answers a question where the answer is readily available. If you figure it out by yourself, fix the documentation, either in the wiki or by providing a patch for the cocoon site.

Last Words

Accept liberal, produce strictly. Put input through tidy etc. Slight mistakes in URLs should still produce sensible results. Missing request parameters should fall back on defaults. Produce validated HTML, WAI standards. Don't fight the separation of concerns. Even a sole developer should separate ruthlessly, you may have to share work or come back to your work after a while.

Please Contribute

The Slides and notes for the talk can be seen here: CocoonBestPractices. Please contribute to the suggestions there.