Best Practises when Developing with Lenya 1.4

Table of contents

1. Speeding Up Development

Following these tips can increase your development productivity:

- Set up your development environment correctly. The time you invest in this will pay off very quickly.
- Practise test-driven development (http://en.wikipedia.org/wiki/Test_driven_development) . Since you can execute the tests with a single click in Eclipse, you don't need to build and restart the application to check if something works.
- During development, set `modules.copy=false` in `local.build.properties`. This way, you don't have to execute the build process when you change something in a module (except Java files and patches for `cocoon.xconf`).
- Put all your non-Java files - XSLTs, CSS files, JX templates and complex sitemaps - in modules. This allows you to change them without rebuilding (see preceding tip). Using modules for anything which is not specific for a publication is a good practise anyway - it encourages generic design and reuse.

2. Building Maintainable Applications

To ensure the simplicity and maintainability of your Lenya-based applications, try the following tips:

- Use resource types sparingly. A new resource type adds complexity to your application. It requires to patch `cocoon.xconf` and is therefore a static element of your application. Most differences between pages can be implemented by using different XSLTs or templates in the presentation layer, or even using distinct samples to provide a starting point for a certain type of page.
- Make extensive use of resource type formats. This way, you create orthogonality - if you implement a certain format in each of your resource types, it is very easy to include arbitrary resources in different locations. Typical examples of formats are print views, teasers, summaries, icons, and RSS feeds.
- Modularize your application. You can put each resource type in a separate module. Service implementations are good module candidates as well - by adding or removing them from your build path you can easily switch between several implementions. A complex application is likely to feature a `shared` module which contains utility XSLTs and other resources which are used across multiple modules.
- Use templates instead of XSLTs for layout purposes. This is an example of the Separation of Concerns (http://en.wikipedia.org/wiki/Separation_of_concerns) paradigm. XHTML templates can be edited without XSLT skills and without influencing your presentation logic. For more information, check out the article Style-free Stylesheets with Cocoon (http://www.cocooncenter.org/articles/stylefree.html) on cocooncenter.org.
- Keep your sitemaps simple. It is very hard to write tests for sitemaps, and complex ones are difficult to read. Instead of building huge nested pipelines (e.g. for error handling), it often makes sense to implement a specific selector or action to handle the complicated aspects of the page flow. As soon as the functionality is implemented in Java, it can be tested using unit tests and refactored using your favorite IDE.
- Don't overload your usecase handler classes with view-specific details. The handler is occupied with business logic, it shouldn't have to deal with the view as well. If the JX templates become too

complex, consider writing utility classes and call them from the template. Modularizing JX templates is another promising approach to reduce their complexity.