How JDepend Changed My Java Packaging

http://chasethedevil.blogspot.com/2006/10/how-jdepend-changed-my-java-packaging.html

One important feature of Java language is the package keyword. It helps a lot in modularizing your code. But how exactly one should use it is not that clear.

I have been on projects where 100s of classes are in a same package and more often where you have packages for every 2 classes. Often, packages are chosen so as to split functionalities. But often as well, people are packages maniacs and create way too many of them, because they want to sort things out, not necessarily applying a consistent logic. For example, I am sure many of you have seen the "blahblah.exceptions" packages where you dump exceptions classes.

Without any precise intention for packages, you encounter quickly cyclic dependencies in your project, and your build system becomes overly complicated. Then you try to remove cyclic dependencies, not by using the package keyword better, but by building jars differently (=ugly jars) and "refactoring" a bit.

I fell myself in many of those traps. Fortunately I stumbled on my way on JDepend package analysis program. At first I wondered why JDepend calculated dependencies between Java packages and not between Jars, as in your project what matters is jar interdependencies. After using it regularly, first for the fun of

trying not to have cyclic dependencies in my Java packages - the green light effect (I wanted my green light on my project), I understood much better its interest. I now make Java packages that are much more meaningful. They not only separate functionalities, but also create excellent modularity. Building a project is very easy, reusing parts of it is very clear. If one needs to split a jar in a project, it's very clear how to split it (by Java packages). And I find that in the end my packages make much more sense.

Placing a class in the right package is most of the time not difficult, especially for experienced programmers. But now and then, with JDepend, I find mistakes. Correcting those mistakes regularly, when it's just about one or two classes is quick and easy. Correcting them on a project where dependency analysis was never done is a nightmare.

Now I don't really care about JDepend metrics about abstractness and instability so much, I did not find any good use of them. But JDepend (or a package dependencies checker) really is an essential program for good Java development.