Acyclic Dependencies Principle

The dependencies between packages must form no cycles.

Principle Foundation

Packages have relationships that are dictated by the classes contained within them, and the relationships that these classes possess with classes in other packages. While time is usually spent designing the relationships between classes, it’s unfortunate that little time is given to the relationships between packages. These package relationships impact reusability, maintainability, and the extensibility of our system.

Packages with few dependencies on other packages are easily reusable, as deployment is much less of a burden. Packages with many dependencies on other packages do not exhibit this same characteristic however, as effort is spent determining the nature of the relations. Packages with cycles in the dependency structure are ill formed because neither package can be used independent of the other.

Sample Illustration

The diagram in figure 1 illustrates compliance and violation of the ADP. At left, we see a client package that contains a Client class. This Client class has a relationship to a Service class in the service package. In this scenario, we can easily reuse the service package and its containing classes, because we know the classes in the service package have no other dependencies. At right, however, we see a cycle in the dependency structure. The Client and Service classes have a bi-directional association, implying that the client package has a relation to the service package, and the service package has a relation to the client package. In this scenario, neither the client nor the service package can be used without the other.

Key Implementation Considerations

- Reuse Impact Packages with cycles in the dependency structure limit reusability. Those packages loosely coupled to other packages have contents that are more reusable.

• Package Coupling  The Acyclic Dependencies Principle emphasizes reducing the coupling that exists between packages. While some coupling must exist, those packages most loosely coupled based upon the contained classes, and subsequent relationships, are more easily maintainable.

• Layering  The Acyclic Dependencies Principle is consistent with how we would layer a system. Typically, upper level layers are dependent on lower level layers. A package should reside in, not span across, a layer. As such, packages in lower level layers should have reduced coupling, increasing the reusability.

Consequences
Foremost, package relationships are often an afterthought when designing systems. That is, if they are given any thought at all. Assuming they are, package relationships only confound the difficulty of design. Not only must the system exhibit high degrees of class resiliency, but they must also exhibit high degrees of package resiliency.

When careful consideration is given to the package relationships, however, it’s usually beneficial to future efforts. Maintenance is eased as new developers are able to see high level views of the system, and understand the high level architectural mechanisms employed much quicker. Carefully designing package relationships lends the luxury of being able to redesign package internals, while knowing exactly the impact of that change.

Related Principles
The Stable Abstractions Principle and Stable Dependencies Principle offer strategies for allocating classes to packages, as well as metrics for determining the stability of our packages. This will impact our package dependency structure.

References
