Subject: Re: (re-)introducing Dependency Injection
Posted by AJM on Sat, 06 Apr 2013 13:33:08 GMT
View Forum Message <> Reply to Message

Follow these links for definitions of dependency, cohesion and coupling.

One reason that I don't use DI is that I prefer to instantiate an object only when I need it and not before. Sometimes my CUSTOMER object may need to access the ADDRESS object, and sometimes it may not. Why instantiate an object that is not going to be used? In some cases there may be a choice of possible classes, and I don't know which one I want until I come to access it. There may also be arguments that I need to supply to the constructor which are only known at the last moment.

The main reason I don't use DI is that it requires too much effort for too little reward, and my code is easier to write without it. It is also easier to read and maintain, especially by others, and that is *FAR* more important than following some stupid principle devised by some DIC head (do you see the play on words there?)

Your point that "main classes aggregating other classes should not depend on the direct implementation of the aggregated classes" is just an opinion as far as I am concerned and not an absolute rule. Unless you can prove that breaking this rule has unwanted consequences then I shall continue breaking it until the cows come home.

Another point which falls into the same category is "both the classes should depend on abstractions". What on earth does that mean? A class is the result of performing an abstraction, so what does "abstraction" mean in that statement? What does the statement "abstractions should not depend on details, details should depend on abstractions" supposed to mean? This sounds like pure gobbledegook to me.

You statement about the menu system being intertwined with the security system is also pointless. There is a menu system and there is a security system as they each have their own separate sets of maintenance screens. They interact at runtime when a list of menu options is filtered to remove those to which the user does *not* have access, thus producing a list which the user *can* access. They are separate, yet they work together to produce a result.

The idea that you may want to swap out one menu system for another is pointless. Radicore has a single built-in menu system which cannot be changed for another. If you don't want to use the Radicore menu system then don't use Radicore.

Finally, you ask the question "how to accomplish this?" How to accomplish what exactly? Your post is so full of numerous meandering statements it is difficult to see where there is a real question.