Liz Brondos
10/28/03

Discussion Between an Angry Maintenance Programmer and his Naïve Assistant

Maintenance Programmer: I've been looking at this code you wrote for the application we're both working on, and, frankly…
Naïve Assistant: Yes?
Maintenance Programmer: Well, it STINKS ! I mean, it doesn't even compile!
Naïve Assistant: Well, I was trying to make some changes, but I guess I didn't make the changes everywhere in the program that I needed to.
Maintenance Programmer: My goodness, don't you know ANYTHING about class structure?
Naïve Assistant: Um…no.
Maintenance Programmer: *(Sighs.)* I need to teach you. Where should I start? Oh yeah, you need to know about coupling. Coupling describes the interconnectedness of classes. The tighter the coupling, the more interconnected the classes are. It is good to have loose coupling though, because with loose coupling, each class is largely independent and communicates with other classes via a small, well-defined space.
Naïve Assistant: What's the benefit of that?
Maintenance Programmer: Well, that way, when you need to make a change for one class, you don't need to go running around like an idiot trying to change the other classes so much. That's what you've been doing.
Naïve Assistant: Oh.
Maintenance Programmer: You have too many public variables! That makes your coupling really tight, and you need to make some of them private in order to loosen the coupling. You need proper encapsulation in classes to reduce coupling, which leads to a better design.
Naïve Assistant: Oh.
Maintenance Programmer: Also, you need to write a more cohesive system. This means that each unit of code, whether it is a method, class or module, needs to be responsible for a well defined task or entity. Your methods are doing two or three completely different things at a time, and that is terrible!
Naïve Assistant: Oh.
Maintenance Programmer: Oh my goodness, look at all your use of code duplication! This is a nightmare! You have the same segment of code in the application twenty zillion times! So when you made a change in this segment of code, you only changed it in one occurrence of this piece of code, and not all twenty zillion!
Naïve Assistant: Oh.
Maintenance Programmer: Not only is your system not cohesive, but you know nothing about responsibility-driven design! That means that you need to design your classes in a way that you assign well-defined responsibilities to each class. You can use this process to determine which class should implement which part of an application function.
Naïve Assistant: Oh.

Maintenance Programmer: One of the most important concepts of good class design is that of localizing change, which means that making changes to one class should have minimal effects on other classes. But your coupling is so tight that changing one class has huge effects on all the other classes!
Naïve Assistant: Oh.
Maintenance Programmer: You need to do some major refactoring! This means you have to restructure your design to maintain a good class design when the application is modified or extended.
Naïve Assistant: Oh.
Maintenance Programmer: You are a moronic, stupid, idiotic, disorganized excuse of a programmer! Your program is a complete piece of trash which I will toss out into the recycling bin! I am never, EVER going to work with you again!
Naïve Assistant: Wait a minute, WHO ARE YOU CALLING TRASH?
(And the violence begins…)