Code Reuse

Steven Bucksbaum, February 5, 2011

Single Source of Failure versus the Copy-And-Paste School

Copy and Paste School: Create duplicate code every time the same code logic is needed in another location.

The Copy and Paste Problem: Any errors in the code you copied are now repeated. When the bug is discovered, there are more places to find and fix the error. Assuming the coder remembers (and the original engineer is probably no longer with the company) where the multiple locations code has been placed.

The Solution: Create a New Routine or Method.
Create a single location for the logic or code to reside. When the need to reuse code is discovered, create a new routine or method for the logic. Put the code in a single place and then call the routine when that code logic is needed.

Techniques For Code Reuse
1. **Inheritance**: Create a parent class for the shared routine. Each child class will have access to shared logic.
2. **Polymorphism**: Use polymorphism to take advantage of shared code in a class library.
3. **Static Methods:** Many routines can be placed in a static method. Static methods can be access universally throughout the code base.

Reasons to promote Code Reuse and Single Source of Failure
1. **Reducing Complexity:** After it is written, you can reuse the routine without detail knowledge of the logic. Other reasons include minimizing code size, improving maintainability, and improving correctness. With the power of abstracting routines, complex programs would be intellectually impossible to manage.
2. **Avoiding Duplicate Code:** Similar code in two routines is warning sign of a design error. David Parnas says to move the logic to its own routine. Future modifications will be easier because you will need to modify the code in only one location. The code will be more reliable because you will have only one place to be fix and test the code.
3. **Limiting the Effects of Change:** Isolate areas that are likely to change so that effects of the changes are limited to a single routine or a few routines. Areas likely to change include hardware dependencies, input/output formats, complex data structures and business rules.
4. **Improving Performance:** Having code in one place means a single optimization benefits all code that calls that routine. And makes it more practical when code re-factoring is required.
5. **Easier Maintenance:** Fix a bug in one place and all who call routine benefit from the improved logic.