**CHECKLIST: Self-Documenting Code**

**Classes**

❑ Does the class’s interface present a consistent abstraction?

❑ Is the class well named, and does its name describe its central purpose?

❑ Does the class’s interface make obvious how you should use the class?

❑ Is the class’s interface abstract enough that you don’t have to think about how its services are implemented? Can you treat the class as a black box?

**Routines**

❑ Does each routine’s name describe exactly what the routine does?

❑ Does each routine perform one well-defined task?

❑ Have all parts of each routine that would benefit from being put into their own routines been put into their own routines?

❑ Is each routine’s interface obvious and clear?

**Data Names**

❑ Are type names descriptive enough to help document data declarations?

❑ Are variables named well?

❑ Are variables used only for the purpose for which they’re named?

❑ Are loop counters given more informative names than *i*, *j*, and *k*?

❑ Are well-named enumerated types used instead of makeshift flags or Boolean variables?

❑ Are named constants used instead of magic numbers or magic strings?

❑ Do naming conventions distinguish among type names, enumerated types, named constants, local variables, class variables, and global variables?

**Data Organization**

❑ Are extra variables used for clarity when needed?

❑ Are references to variables close together?

❑ Are data types simple so that they minimize complexity?

❑ Is complicated data accessed through abstract access routines (abstract data types)?

**Control**

❑ Is the nominal path through the code clear?

❑ Are related statements grouped together?