3.1 Testing
One needs to test extensively. Start by trying some standard simple data. Look at the boundary values: make sure it handles the smallest or largest value the program must work for, and suitably rejects the value just out of range. Add watches or debug statements so that you know what is happening at all times. Especially look at the empty case, or the 0 input.

3.2 Avoiding Problems
A function should normally check its arguments. It notifies the caller of a problem by using an exception (discussed later) or a special return value. However, the programmer should try to avoid exceptions: consider error recovery and avoidance.

3.3 Literate Programming
Good programming requires extensive comments and documentation. At least:

explain the purpose of each instance variable, and for each method explain its purpose, parameters, returns, where applicable.

You should also strive for a consistent layout and for expressive variable names. For a class, one might list the functions, constructors and public fields, and for each method explains what it does together with pre-conditions, post-conditions, the meaning of the parameters, exceptions that may be thrown and other things.

UML is an extensive language for modeling programs especially those for an object-oriented programming language. It is a system of diagrams designed to capture objects, interaction between objects, and organization of objects, and then some.

3.4 Algorithms
An algorithm for a problem is a recipe that:
(a) is correct,
(b) is concrete,
(c) is unambiguous,
(d) has a finite description, and
(e) terminates.

Having found an algorithm, one should look for an efficient algorithm. As Shaffer writes: “First tune the algorithm, then tune the code.”