Refactoring methods future proofs them by improving comprehension and intent.

Predictable code:

* Add guards to check incoming params and only open the gates to valid or expected input. **Guard clauses** are statements that check param validity.
* Minimise the number of params. The fewer the better because there are less to validate, less chance for introducing bugs and easier to test. If you have many related params, consider making a class.
* Consistent param ordering. Most important to least important. By remaining consistent you minimise the chance of bugs over time say if params are added or taken away.

Decimal.TryParse – determines if it’s a number

Convert.ToDecimal

Fail Fast technique – causes the code to fail immediately when it receives something invalid.

String.IsNullOrWhiteSpace

Method overloading allows you to clean up methods. For example refactoring a method into 2 with the 1st method possibly doing some type conversion before passing it to the overload which has the correct type.

Design by contract – each method must abide by a contract that answers 3 questions:

* What does the method expect? Acceptable and unacceptable inputs.
* What does the method guarantee? Promised return value and expected error or exceptions
* What does the method maintain? Defining any properties used or set by the method