TDD usage patterns

Red bar

1. Step by step – from the list of test cases choose the one you're most sure of
2. First test – the one which is the simplest, requires least setup
3. Explanatory tests – for everything that needs from you some thinking find examples and code them as tests
4. Learning tests – if there's some library / API to learn, do it by writing tests. You'll be sure you understand it well.
5. Regression test – once you know your code has an error, write the smalles test that proves the code wrong. Once the code's fixed, you'll have a proof it's correct now.
6. Start anew – the step was too big? You're lost? Oftentimes it's good to remove everything and start from scratch.

Testing

1. Test-child – if you have a test hard to write, write a smaller one containing something that failed in the big one. Once this one's working it'll be easier to get back to the big one.
2. Failing test – every coding session finish with a failing test. When you're back to this topic, it'll remind you where you finished the last time.
3. Clean commit – always commit only passing tests – you're not alone in the universe!

Green bar

1. Triangularization – introduce abstractions only when you've got at least 2 tests.
2. Obvious implementation – if you know how to implement something, don't pretend you don't.
3. Join the differences – if you want to unify two similar pieces of code, start with differences. Eliminate them by introducing abstractions one by one, until the code gets identical. Then remove the duplication.
4. Isolate changes – if you're supposed to change some code, first isolate it to make sure changing it won't break something else.