Test coverage has to be increased

But when writing tests:

- Make sure quality of the tests is also high, otherwise these tests are not more than useless mess.
- Keep them as clean and as small as possible. Especially for unit tests, there is no reason to have a unit test 1 page long.
- Chose a proper category is necessary:
    - **Unit Test** - no category specified. In this case test must be:
        - small
        - very fast
        - should not use any external resources except class being tested, otherwise mock them, if it takes too much to mock - refactor both class and test!
    - **[Category("Integration")]** - when you need to test many classes working together, or even whole system.
    - **[Category("DataAccess")]** - the same as an **Integration** but involving any kind of reading/writing, like import, save project file, etc.
    - **[Category("Performance")]** - test which checks if some code does not run slower than expected. Keep code beteen stopwatch.Start() stopwatch.Stop() as small as possible, no timing of log.Debug() or any other crap.
    - **[Category("Windows.Forms")]** - put this category if you use WindowsFormsTestHelper.Show(Control control) to visualize any form. This is actually another type of Integration test. It mainly tests if controls, windows can show correctly. If possible - do some operation via API and add Asserts there as well.

Put [Ignore("message")] on those tests which are not ready yet (we have to clean-up them, there are too many already, they should have very limited lifetime.

Some rules from uclebob:

- You are not allowed to write any production code unless it is to make a failing unit test pass.
- You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
- You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

This is how our test coverage looks like today: