# Lab 2. Unit testing

## Lab goals:

1. Unit tests creation in order to test software components functionality.
2. Tests generation using NUnit, Unit Test.

## Lab work tasks:

1. Generate software unit tests in order to evaluate software quality for chosen software. The JTest, C++ Test or dot Tests programs could be used. Create some unit tests manually.

We generated test for the patterns that are used in the game. Patterns that were included: singleton, factory, abstract factory, builder, strategy, command, observer, adapter, façade.

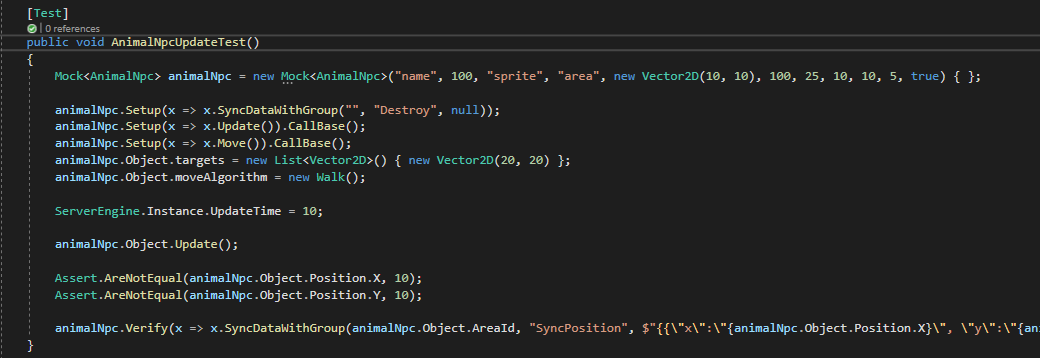
Classes that were excluded: **network manager, program, startup, chathub**. They were all excluded because they use network connections.

1. Generate tests for a whole software application (100% code coverage)

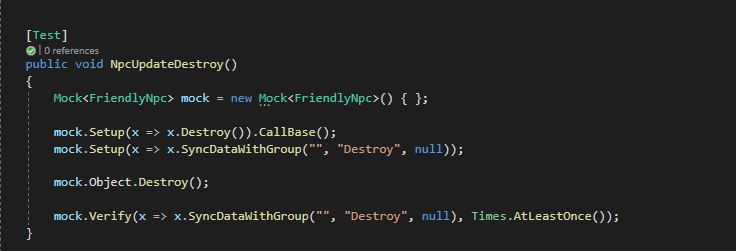
//reikia nuotraukos

We couldn’t reach the 100% coverage, because there were several scripts excluded.

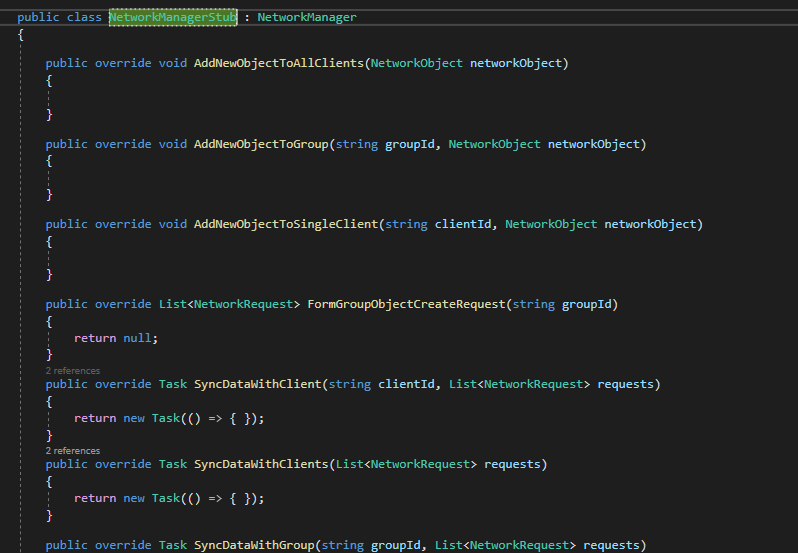
1. Research mocks, stubs, drivers, use them while creating unit tests where appropriate.



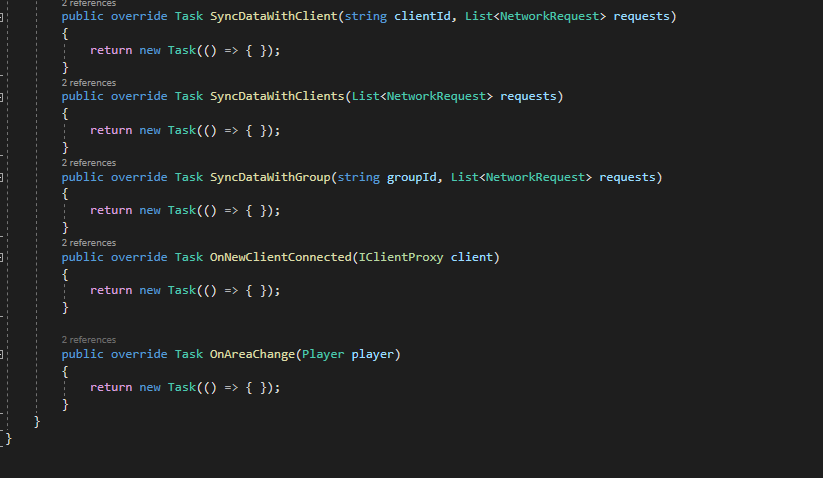
Code , mock



Code , mock



Code , stub



Code , stub

1. Research parametrized tests, use them while creating unit tests where appropriate.

One of the places where we used parametrized tests is command pattern, because it let’s us know if the commands are working properly and if it’s possible to undo them correctly.

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

Code , parametrized test

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

Code , parametrized test

Paveikslėlis, kuriame yra žinutė

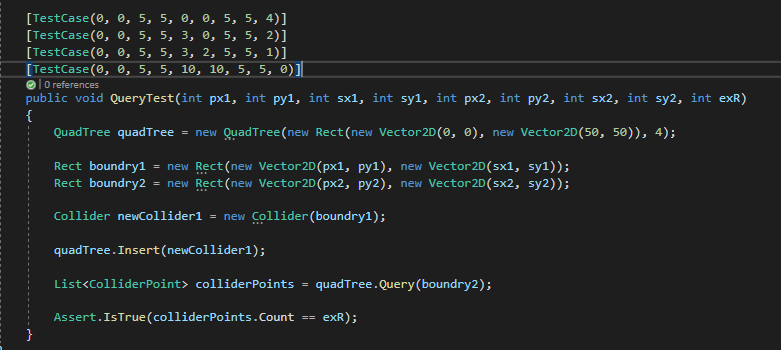
Automatiškai sugeneruotas aprašymas

Code , parametrized test

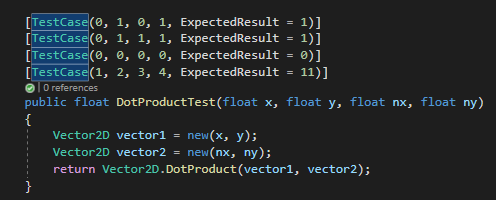
Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

Code , parametrized test

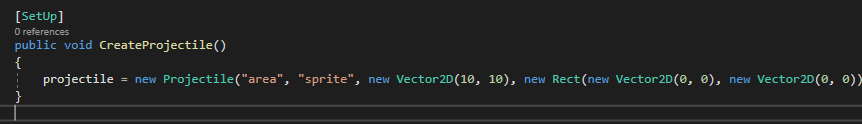


Code , parametrized test



Code , parametrized test

1. Research tests set-up, tear-down phases, use them while creating unit tests where appropriate.



Code , set-up

1. Create tests using graphical editor for one chosen class to test.
   1. Our chosen class is **Vector2D.** To create an unit test for the whole class, you have to right-click on the class and select “Create Unit Tests”.

Paveikslėlis, kuriame yra žinutė

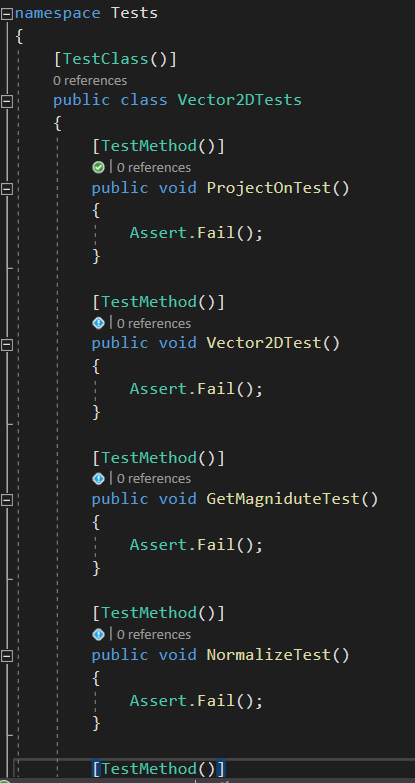
Automatiškai sugeneruotas aprašymas

* 1. Select the test project and an output file name

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

* 1. NUnit automatically generated code for all of the methods



* 1. The methods that were generated are empty, that’s why we have to fill in them ourselves. This is an example unit code for the **Vector2D.ProjectOn()** method.

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

1. Create unit tests in code for one chosen class to test.
   1. First we need to create a new class for testing

Paveikslėlis, kuriame yra žinutė, ekrano nuotrauka, monitorius, juoda

Automatiškai sugeneruotas aprašymas

Paveikslėlis, kuriame yra žinutė, monitorius, ekrano nuotrauka, ekranas

Automatiškai sugeneruotas aprašymas

* 1. Then we add the necessary attributes to the class and methods.

Paveikslėlis, kuriame yra žinutė

Automatiškai sugeneruotas aprašymas

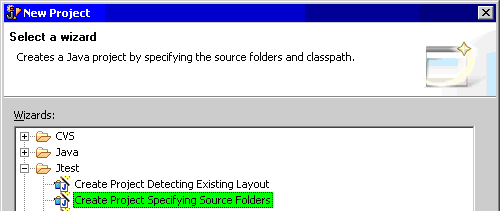
* 1. Last but not least, we add the testing code for the class methods



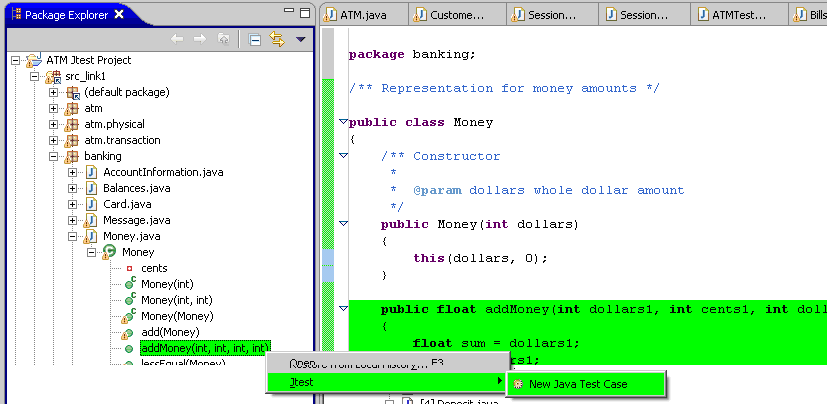
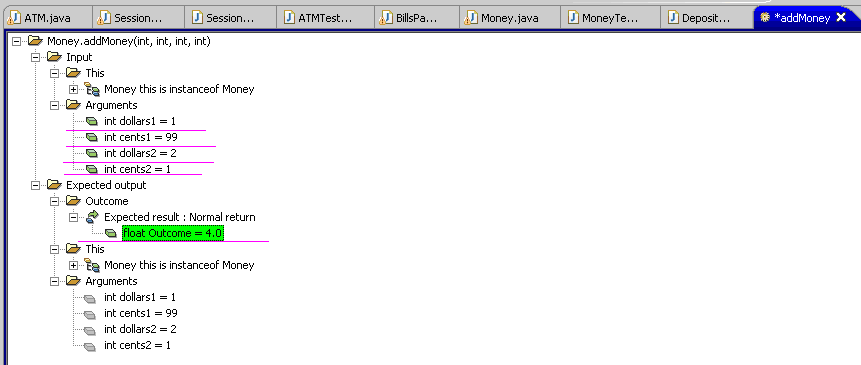
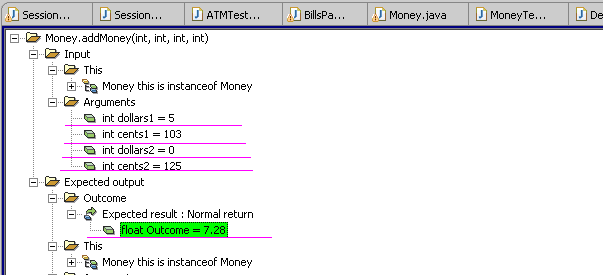
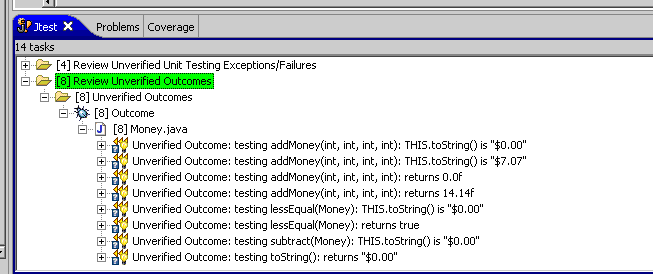
1. Evaluate software tests coverage.

//reikia nuotraukos

## Step by step instructions:

1. Launching the testing tool
   1. Launch Parasoft JTest tool from the start menu if software under test is implemented in Java. Parasoft C++ test can be used if software under test is implemented in C++, and Parasoft dot Test for software that is implemented in one of .NET programming languages.
2. Importing software under test.
   1. Use File-> import for importing software project into tool for testing.
   2. Or create a new Java project and copy source code to the project.
3. Creating test project.
   1. Use File->new Project
   2. 
   3. Select JTest → Create Project Specifying source folders
   4. Give a project name (ex: text project123)
   5. Specify the created project location in the following window.
   6. Complete test project creation by selecting “Finish”
4. Adding fake class for testing (only for this example)
   1. Create class Money (Context menu on project -> new -> class)
   2. Add the method to the class:

|  |
| --- |
| public float addMoney(int dollars1, int cents1, int dollars2, int cents2)  {  float sum = dollars1;  sum += dollars1;    float cents = cents1 + cents2;    if (cents > 100)  {  sum += 1;  }    sum += (cents % 100) / 100.f;    return sum;  } |

1. Automatic tests generation
   1. Select imported/created software under test project.
   2. Use JTest->Test using example configuration from context menu on the project.
   3. Check the generated tests count in “Generation” window.
   4. Check executed tests count in “Stats” window.
   5. Then tests generation and execution is completed, check testing report by clicking “Report”
2. Visual test creation:
   1. On step there was created a class with some strange code that looks like an invitation for bugs.
   2. Select Money class method “addMoney” in the package explorer.
   3. Invoke JTest-> New Java Test case from context menu.
   4. 
   5. Accept default in the opened window and click “Finish”
   6. Fill the test data and expected result in a new windows as presented in the following picture:
   7. 
   8. Add yet another test in the same way:
   9. 
   10. Execute create test in the same way as in 5-th step, observe the testing report.
3. Executing tests for a single class:
   1. In some cases it’s not convenient to execute all tests at once.
   2. Select class Money in package explorer.
   3. Invoke „JTest → Test Using example configuration” from context menu.
   4. In this case the generated and crated tests are executed the ones that are related to the Money class.
   5. The testing results should be similar the results in the following figure:
   6. 
   7. Find tree node “unverified outcomes” in JTest window. It’s time to update and manually check if all test outcomes are correct. If returned value is correct select verified option by selecting “Quick fix” and checking “Verify outcome” then the outcome has a correct value. Then outcome value is incorrect, the bug has been discovered. It case the correct outcome value has to be set, and code with a bug fixed (in money class).
4. Unit test creation by implementing test code:
   1. Create class MoneyTest2 in test project.
   2. Add the following test code to the created test class:

|  |
| --- |
| @Test  public void testAddMoneyHandMade() throws [Throwable](http://www.google.com/search?hl=en&q=allinurl%3Athrowable+java.sun.com&btnI=I%27m%20Feeling%20Lucky)  {  Money THIS = new Money();  // jtest\_tested\_method  float RETVAL = THIS.addMoney(3, 1, 101, 102);  assertEquals(5.04f, RETVAL, 0.0f); // Verified by Administrateur  // No exception thrown  // jtest\_unverified  } |

* 1. Execute created test in the same way as in 5-th step.
  2. Observe resting results in JTest window.
  3. Fix class Money until all tests pass.