**Test Plan Document**

**for**

**Producers-Consumers Economic Relationships Model**

**Version 1.0**

**Prepared by Mateusz Golab**

**Cranfield University**

**School of Engineering**

**AMAC**

**1st April 2012**

**Table of Contents**

[1. Introduction 3](#_Toc321873636)

[1.1. Objectives 3](#_Toc321873637)

[1.2. Testing process overview 3](#_Toc321873638)

[1.3. References 4](#_Toc321873639)

[1.4. Glossary 4](#_Toc321873640)

[2. Tools 4](#_Toc321873641)

[3. Test items 5](#_Toc321873642)

[3.1. Program modules 5](#_Toc321873643)

[3.2. Features to be tested 5](#_Toc321873644)

[3.3. Features not to be tested 5](#_Toc321873645)

[4. Testing strategy 5](#_Toc321873646)

[4.1. White Box Testing 5](#_Toc321873647)

[4.2. Black Box Testing 6](#_Toc321873648)

[5. Testing Process 6](#_Toc321873649)

[5.1. Unit Testing 7](#_Toc321873650)

[5.2. Integration Testing 7](#_Toc321873651)

[5.3. Acceptance Testing 7](#_Toc321873652)

[5.3.1. Performance testing 7](#_Toc321873653)

[5.3.2. Trace – ability matrix 7](#_Toc321873654)

[6. Items Pass / Fail Criteria 8](#_Toc321873655)

[6.1. Unit Testing 8](#_Toc321873656)

[6.1.1. Pre-requisites 8](#_Toc321873657)

[6.1.2. Pass criteria 9](#_Toc321873658)

[6.1.3. Fail criteria 9](#_Toc321873659)

[6.2. Integration Testing 9](#_Toc321873660)

[6.2.1. Pre-requisites 9](#_Toc321873661)

[6.2.2. Pass criteria 9](#_Toc321873662)

[6.2.3 Fail criteria 9](#_Toc321873663)

[6.3. Acceptance Testing 10](#_Toc321873664)

[6.3.1. Pre-requisites 10](#_Toc321873665)

[6.3.2. Pass criteria 10](#_Toc321873666)

[6.3.3. Fail criteria 10](#_Toc321873667)

[7. Testing Process Results 10](#_Toc321873668)

[7.1. White Box Testing 10](#_Toc321873669)

[7.1.1 Consumer 11](#_Toc321873670)

[7.1.2. Factory 12](#_Toc321873671)

[7.1.3. Model 13](#_Toc321873672)

[7.1.4. Offer 14](#_Toc321873673)

[7.1.5. Order 14](#_Toc321873674)

[7.1.6. Product 15](#_Toc321873675)

[7.1.7. Producer 16](#_Toc321873676)

[7.1.8. Simulation Manager 18](#_Toc321873677)

[7.1.9. Simulation Presenter 19](#_Toc321873678)

[7.2. Black Box Testing 19](#_Toc321873679)

[8. Testing Deliverables 19](#_Toc321873680)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for changes** | **Version** |
| Mateusz Golab | 1/04/2012 | Document created | 1.0 |
|  |  |  |  |

# Introduction

This document is the Test Plan for the Producers-Consumers Economic Relationships Model. The aim of this document is to describe testing strategy and approach used to verify produced software. Following software quality attributes are taken into account in verification process

* Availability
* Correctness
* Reliability
* Testability

## Objectives

The purpose of this document is to :

* Define the activities required to prepare and perform unit, integration and acceptance testing.
* Describe and follow employed testing strategy
* Define testing tools used
* Define deliverables

## Testing process overview

The testing activities performed to ensure quality attributes of produced software.

1. Create test harness.
2. Implement unit tests for every class.
3. Execute tests using test harness.
4. Fix software bugs to achieve all unit tests passed.
5. Implement integration tests.
6. Execute tests using harness.
7. Fix software bugs to achieve all tests passed.
8. Generate code coverage report.
9. Prepare acceptance tests
10. Perform acceptance tests.

## References

[1] Advanced Software Engineering Lecture Notes, Academic Year 2010/2011 , Dr. Stuart Barnes

[2] IEEE Standard for Software Test Documentation (Std 829-2008)

[3] IEEE Standard for Software Verification and Validation (Std 1012-2004)

[4] <http://namgivu.wordpress.com/2010/10/14/how-to-get-unit-testing-code-coverage-run-from-command-line/>

## Glossary

Available in *Software Requirements Specification for Producers – Consumers Economic Relationships Model* document .

# Tools

Two testing tools are used together with Integrated Development Environment to enable testing process for this project.

* **Google C++ Testing Framework**

Testing framework which provides functionality of unit testing . It can be easily integrated with Visual Studio IDE .

* **Microsoft Visual Studio 2010 Performance Tools**

Profiling and testing tools provided with Visual Studio 2010 IDE. These tools are used to provide code coverage of developed unit tests.

* **Vsinstr**

Used to perform coverage instrumentation of unit testing binary. Executed with following command :

*Vsintr Test.exe /COVERAGE .*

* **VSPerfMon**

Tool used to perform code coverage collection monitoring. Generates code coverage report. Executed with following command :

*VSPerfMon.exe /COVERAGE /OUTPUT:codeCoverageReport*

* **Microsoft Visual Studio 2010 IDE**

All unit tests are implemented and run using this environment together with Google Testing Framework. This IDE also gives possibility of presenting code coverage reports generated with VSPerfMon.

# Test items

## 3.1. Program modules

Implementation of the project comprises of the following classes :

* Consumer
* Factory
* Model
* Offer
* Order
* Producer
* Product
* SimulationManager
* SimulationPresenter

## 3.2. Features to be tested

All modules of the program are tested with unit, integration and acceptance tests . It is necessary to ensure quality , completeness, correctness, reliability and availability of all modules and entire program. All requirements and quality attributes included in Software Requirements Specification should be fulfilled .

## 3.3. Features not to be tested

The feature which is not tested in produced software is user interface usability. Advanced user interface is not required for this project. What is more, user interface is limited to console interface, so usability of the software from the User point of view is not checked.

# Testing strategy

Testing strategy includes White Box Testing and Black Box Testing.

## White Box Testing

The White Box Testing strategy focuses on internal perspective of the software. This software testing methodology is applied to perform unit and integration tests. At the beginning of testing process, unit tests are implemented for all classes mentioned in chapter 3. Next step is testing at integration level. The quality of White Box testing can be measured in code coverage percentage. It shows how many paths of the software were visited by written tests.

## Black Box Testing

The Black Box Testing is a method that tests software functionalities included in specification. This method is used to perform acceptance testing of *Producers-Consumers Economic Relationship Model*.

# Testing Process

Testing proces

## Unit Testing

The purpose of this phase is to ensure that the class logic is complete and correct ,what ensures that component works as designed. Unit testing is the most low-level part of testing process. Testing each method of every class was important in terms of high code coverage percentage.

## Integration Testing

This phase of testing process checks correctness and completeness of modules relationships. Integration tests were implemented in the same manner as unit tests, except operating on many modules within a single test. Testing relationships between modules was essential to achieve code coverage above 95% in each class . Performing operations involving many modules dependent on each other is the only way to check all possible paths in a software.

## Acceptance Testing

In Acceptance test phase Black Box tests are performed to check completeness and correctness of software functionalities. All functionalities included in requirements specification should be checked by executing software with particular scenario.

### Performance testing

Performance tests are performed at acceptance level. Every time Black Box test is executed software performance is checked. According to the specification produced software should work in real-time, so there should not be any lags during running program.

### Trace – ability matrix

Creating Trace-ability matrix shows whether all functional requirements are covered by use cases. Such situation takes place in this project. It is possible to see below, that all functional requirements are covered by use-cases. Descriptions of functional requirements and use cases are available in *Software Requirements Specification* document.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Use  Case | **1** | **2** | **3** | **3.1** | **3.2** | **3.3** | **3,4** | **3,5** | **3.6** | **3.7** | **3.8** | **3.9** | **3.10** | **3.11** | **4.1** | **4.2** | **4.3** | **4.4** | **4.5** | **4.6** | **4.7** | **4.8** | **5.1** | **5.2** | **5.3** | **5.4** | **6.1** | **6.2** | **6.3** | **6.4** | **7.1** | **7.2** | **7.3** | **7.4** | **7.5** | **7.6** |
| FR-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-2.9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-6.11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-7.1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-7.2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-7.3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-7.4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-7.5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FR-7.6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Trace - ability matrix

# Items Pass / Fail Criteria

## Unit Testing

### Pre-requisites

* Implementation of all classes is completed.
* Test Harness created.
* All unit tests implemented and added to test harness

### Pass criteria

* Test harness executed correctly
* All tests passed

### Fail criteria

* Test harness execution failed
* Test harness executed with fails.

## Integration Testing

### Pre-requisites

* Test harness executed correctly on unit level
* All unit tests passed
* All integration tests implemented and added to test harness
* Code coverage tools prepared

### Pass criteria

* Test harness executed correctly
* All tests passed
* Overall code coverage above 90%

### 6.2.3 Fail criteria

* Test harness execution failed
* Test harness executed with fails
* Overall code coverage below 90%

## Acceptance Testing

### Pre-requisites

* All integration tests passed
* Full understanding of software requirements achieved.

### Pass criteria

* All functionalities included in *Software Requirements Specification* fulfilled.
* Software performance is satisfying.
* Each Black Box test performed resulted with correct output of the software.

### Fail criteria

* Missing functionality
* Software performance is not satisfying
* Incorrect or unexpected software’s output achieved when running Black Box test

# Testing Process Results

## White Box Testing

There are 58 unit and integration tests divided into 9 test cases. Each test case represents single class tests. All tests passed as presented below.

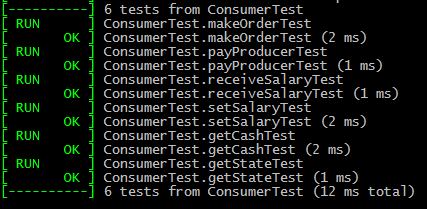




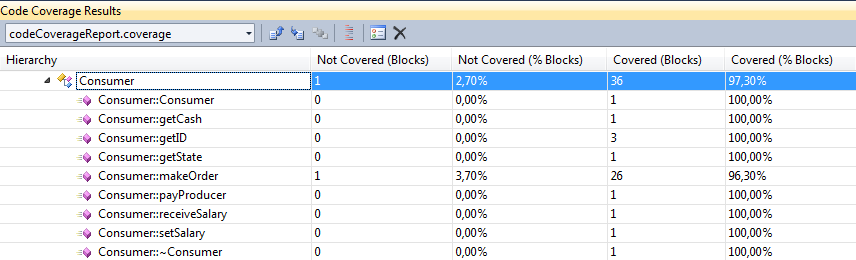
In terms of code coverage there are 685 blocks in software. 673 were covered by unit and integrations tests and 12 were not, what gives overall code coverage result = 98,25 %

### 7.1.1 Consumer

**Tests results :**



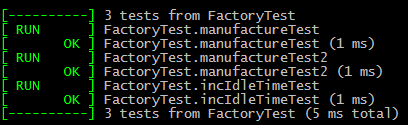
**Code coverage results :**

****

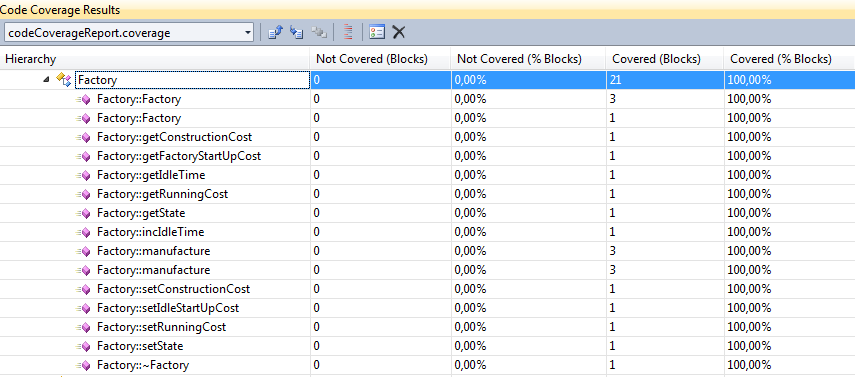
Code coverage for Consumer class is 97,3 %. As we can see one block (path) was not covered by tests in *Consumer::makeOrder* method. This path occurs only in specific situation which requires many cycles to happen and it is difficult to achieve this state from the beginning of simulation. However, 97,3 % is more than 90% which is usually a minimal required value for code coverage at software companies, so this result is satisfying.

### 7.1.2. Factory

**Test results :**

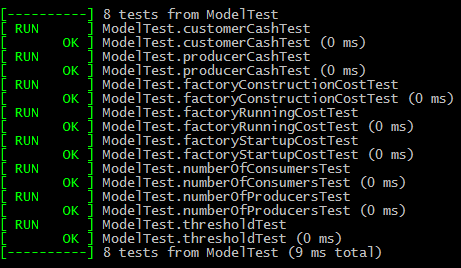


**Code coverage results :**

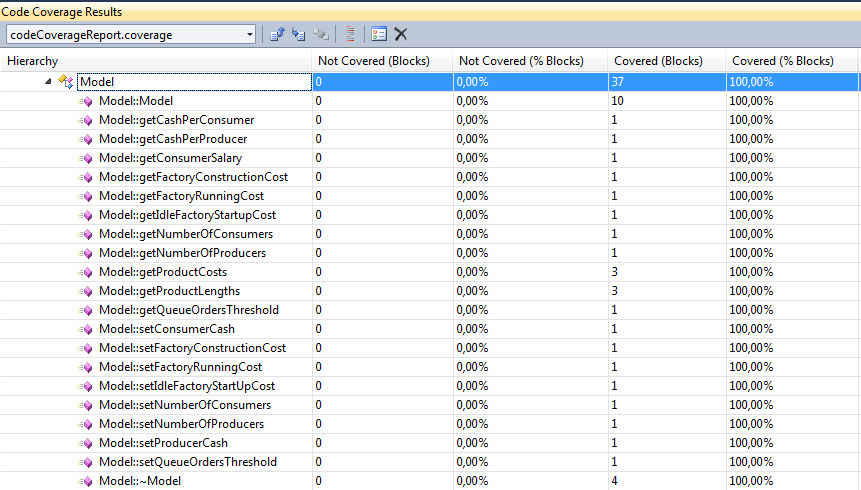
****

### 7.1.3. Model

**Tests results :**

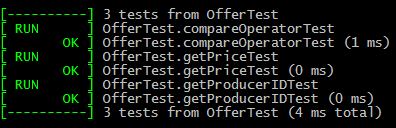
****

**Code coverage results :**

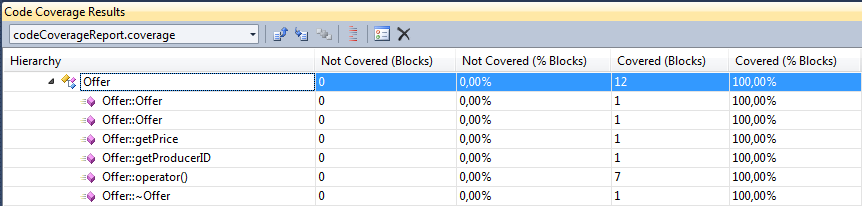
****

### 7.1.4. Offer

**Tests results :**

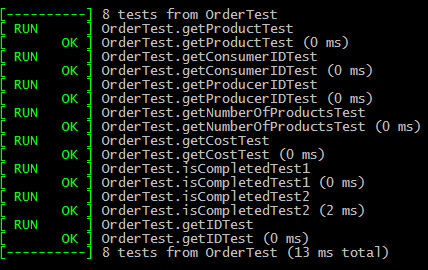
****

**Code coverage results :**

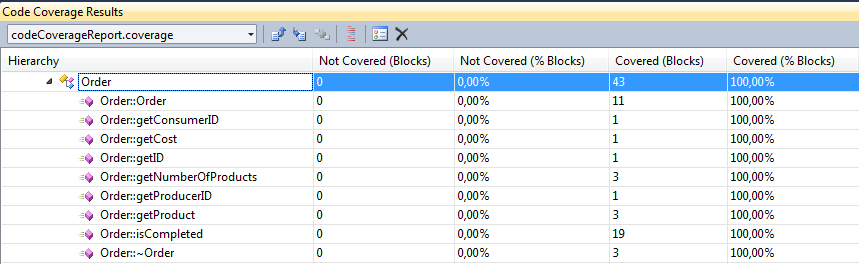
****

### 7.1.5. Order

**Tests results :**

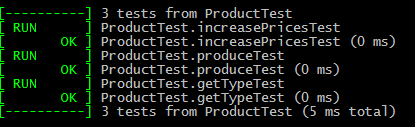
****

**Code coverage results :**

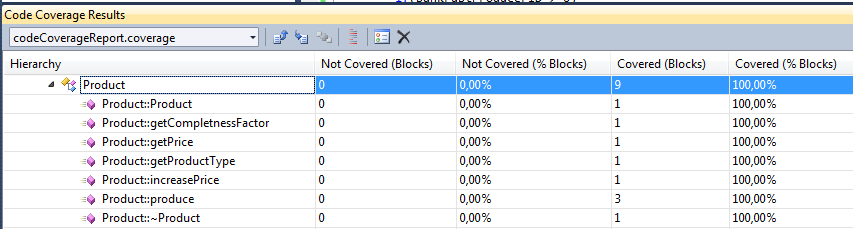
****

### 7.1.6. Product

**Tests results :**

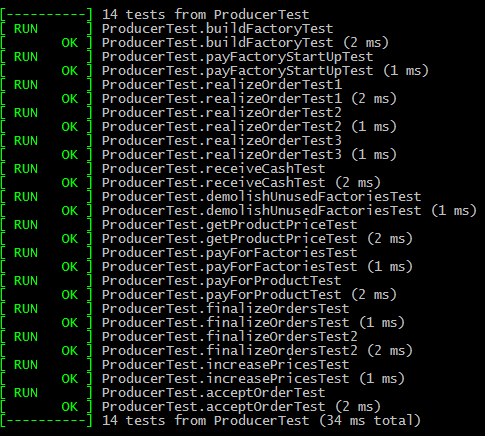
****

**Code coverage results :**

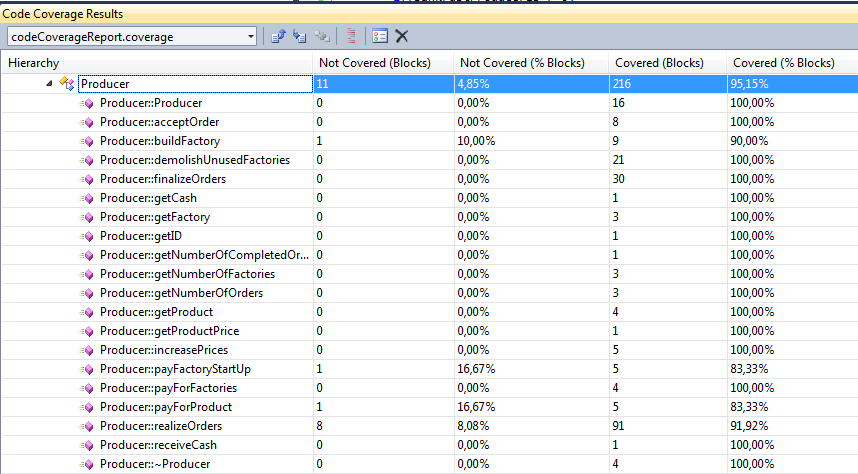
****

### 7.1.7. Producer

**Tests results :**

****

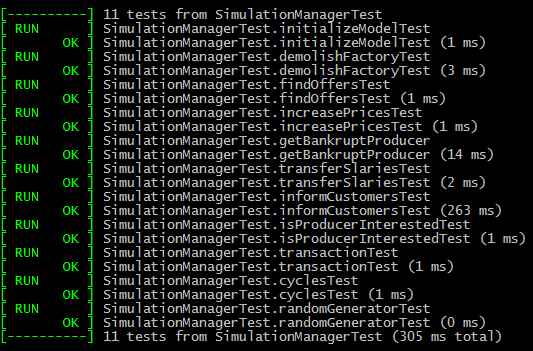
**Code coverage results :**

****

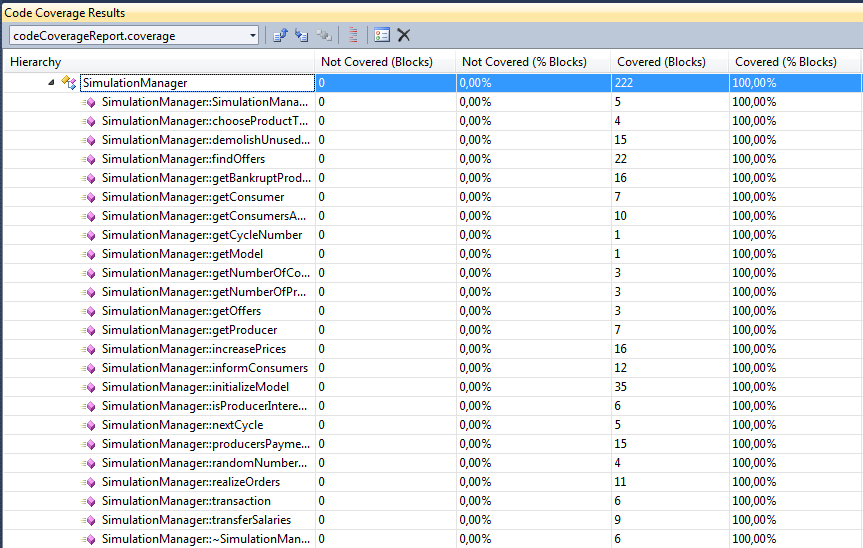
Similarly to *Consumer*  class it was hard to achieve 100% code coverage for *Producer* class due to difficulties with achieving particular simulation state. Missing paths in following methods : *buildFactory, payFactoryStartUp* and  *payForProduct* exist when Producer runs out of cash. With regard to *realizeOrders* the 8 missing paths concern situation which can occur after many cycles and requires a lot of events. Nevertheless 95,15% code coverage is a reasonable result.

### 7.1.8. Simulation Manager

**Tests results :**

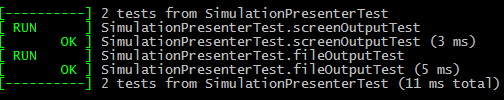
****

**Code coverage results :**

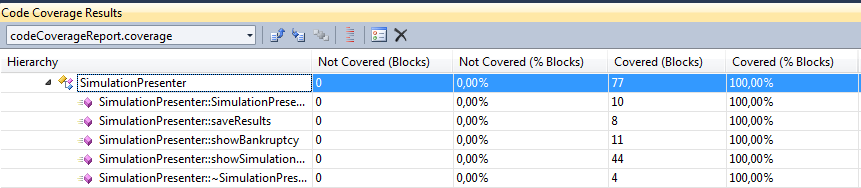
****

### 7.1.9. Simulation Presenter

**Tests results :**

****

**Code coverage results :**

****

## Black Box Testing

There are 3 Black Box tests prepared. Each test was a particular scenario performed on the program. First test was to run simulation with default parameters. Second and third test run simulation after setting several parameters. All scenarios resulted correct output and were achieved in reasonable time, so performance requirements were satisfied.

# Testing Deliverables

* Test Plan document
* Test harness
* Code coverage report
* Tests source codes
* Acceptance testing outputs