

MINISTRY OF SCIENCE AND EDUCATION  
NATIONAL TECHNICAL UNIVERSITY  
“KHARKIV POLYTECHNIC INSTITUTE”  
DEPARTMENT OF SOFTWARE ENGINEERING AND MANAGEMENT  
INFORMATION TECHNOLOGIES

METHODICAL RECOMMENDATION TO  
SOFTWARE ENGINEERING  
LABORATORY PRACTICE

Kharkiv

2016

## Content

Lab № 1 .....	3
Lab № 2 .....	10
Lab № 3 .....	14
Lab № 4 .....	23

## Lab № 1

**Goal: Learning Use Case and Activity Diagrams by using Visual paradigm**

**Tasks:**

1. Get the task and analyze the computational algorithms.
2. Develop the Activity diagram.
3. Develop the Use Case diagram.
4. Prepare the report of the work

Progress of the lab.

### 1. Get the task and analyze the computational algorithms.

According to the task in Table 1 student should analyze computational algorithms.

**Table 1 – Task for lab.№1**

1	$y = \begin{cases} \sum_{i=1}^n (i+x)^2, & x < 0 \\ \sum_{i=0}^{n-1} \prod_{j=1}^n \frac{x+i}{i-j}, & x \geq 0 \end{cases}$	5	$y = \begin{cases} \sum_{i=2}^{n-1} \frac{x}{i}, & x \leq 0 \\ \sum_{i=0}^{n-1} \sum_{j=0}^i \frac{i}{j+x}, & x > 0 \end{cases}$
2	$y = \begin{cases} \sum_{i=1}^{n-1} \sum_{j=1}^n (x-i+j), & x < 0 \\ \sum_{i=0}^{n-1} \frac{x}{i}, & x \geq 0 \end{cases}$	6	$y = \begin{cases} \sum_{i=1}^n \sum_{j=1}^n \frac{1}{x-i-j}, & x < 0 \\ \prod_{i=0}^{n-3} (x-i), & x \geq 0 \end{cases}$
3	$y = \begin{cases} \prod_{j=2}^{n-2} (j+1), & x < 0 \\ \sum_{i=0}^{n-1} \prod_{j=0}^{n-1} (x+i+j^2), & x \geq 0 \end{cases}$	7	$y = \begin{cases} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \frac{1}{x-i+j}, & x \leq 0 \\ \prod_{i=1}^n \left( \frac{1}{x} - \frac{1}{i} \right), & x > 0 \end{cases}$
4	$y = \begin{cases} \prod_{i=0}^{n-1} (i^2 + i), & x \leq 0 \\ \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \frac{x}{i+j}, & x > 0 \end{cases}$	8	$y = \begin{cases} \sum_{i=0}^n \prod_{j=i}^{n-1} (i^2 + j), & x < 0 \\ \sum_{i=1}^{n-2} (i-x), & x \geq 0 \end{cases}$

Author: Melnik K.V., PhD assistant

Department of software engineering and management information technologies, NTU “KhPI”  
Kharkiv, 2016

9	$y = \begin{cases} \sum_{i=1}^{n-1} \sum_{j=1}^n \frac{j}{j^2 + j}, x < 0 \\ x - \sum_{i=1}^{n-1} i, x \geq 0 \end{cases}$	11	$y = \begin{cases} \prod_{i=0}^{n-1} (i^3 + i), x \leq 0 \\ \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \frac{x}{i+j}, x > 0 \end{cases}$
10	$y = \begin{cases} \sum_{i=0}^n (x-i)^2, x \leq 0 \\ \prod_{i=1}^n \prod_{j=0}^{n-1} (x-i-j), x > 0 \end{cases}$	12	$y = \begin{cases} \prod_{j=2}^{n-2} j^2, x < 0 \\ \sum_{i=0}^{n-1} \prod_{j=0}^{n-1} (x+i^2+j), x \geq 0 \end{cases}$

For example, we obtained the following task:

*The application has to calculate the following formula:  $S = \sum_{i=1}^n a \cdot i^2$ . The application should also provide the data input from keyboard and file and output to screen and file.*

As we see in expression, we have unknown variables  $a$  and  $n$ . To calculate the formula student should find the values of  $a$  and  $n$  first.

Let  $a = 5$ ;  $n = 4$ . Than we obtain:

$$S = \sum_{i=1}^n a \cdot i^2 = \sum_{i=1}^4 5 \cdot i^2 = 5 \cdot 1^2 + 5 \cdot 2^2 + 5 \cdot 3^2 + 5 \cdot 4^2$$

## 2. Develop the Activity diagram.

The free specialized software **Visual Paradigm Community Edition** [<https://www.visual-paradigm.com/>] is recommended to draw the Activity diagram (Figure 1.1 - 1.2).

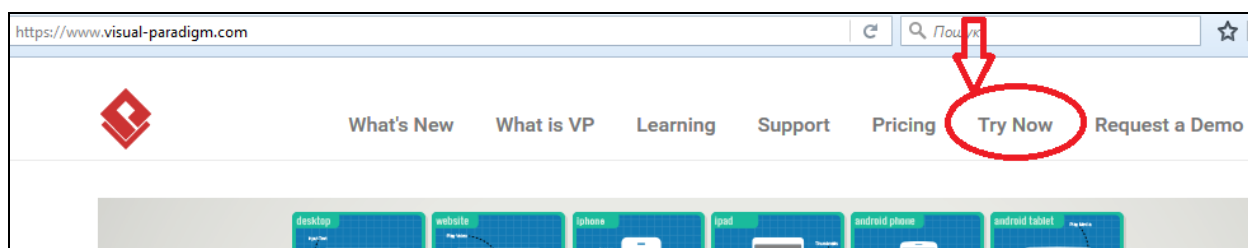


Figure 1.1 – Visual Paradigm main menu

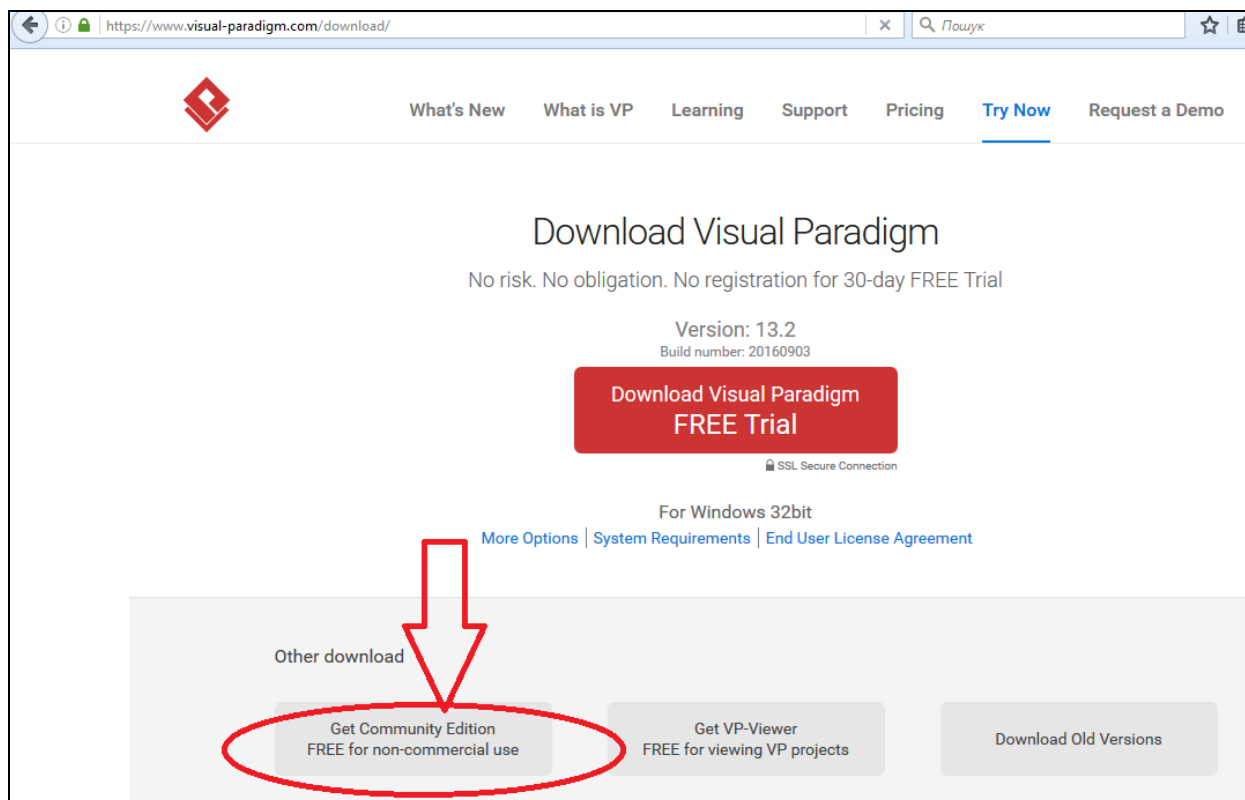


Figure 1.2 – Page with different versions of Visual Paradigm

Start Visual Paradigm. Create new project using menu: “File → New Project”. Define the name of project. Then create activity diagram and set the name using menu: “File → New Diagram → New Activity Diagram”; or click the “Activity Diagram” menu in the Diagram Navigator and then select the “New Activity Diagram”.

**Activity diagram** is used to describe dynamical properties of the system. For example, it is possible to use it to describe different algorithms that are necessary to implement in the system. The following nodes and edges are typically drawn on UML activity diagrams (Figure 1.3): activity, partition, action, object, control, activity edge.

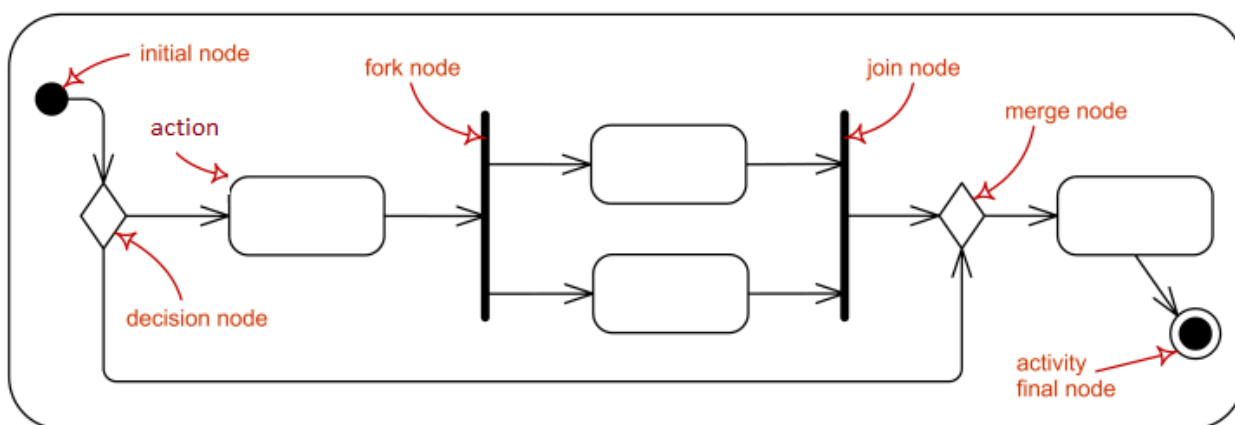


Figure 1.3 – Basic Activity Diagram elements

In our case the computational algorithm should look like shown on Figure 1.4.

There is possibility of changing font value in an activity diagram: right click an element or a diagram, then in “Styles and Formatting” menu student should choose “Formats” menu. User can set the diagram transparency the same way: “Styles and Formatting → Transparent”.

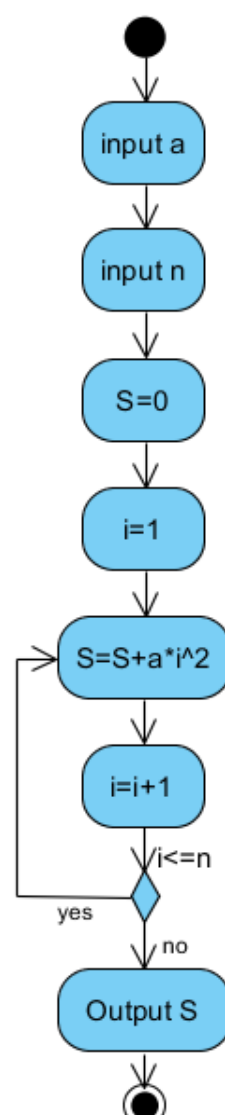




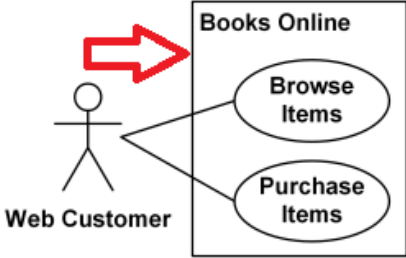
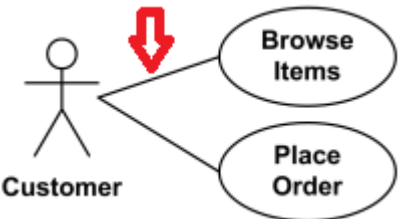
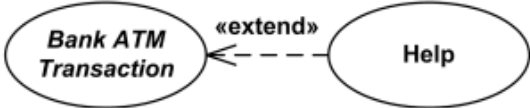
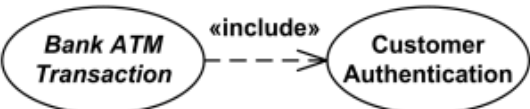
Figure 1.4 –Activity Diagram  
for chosen task

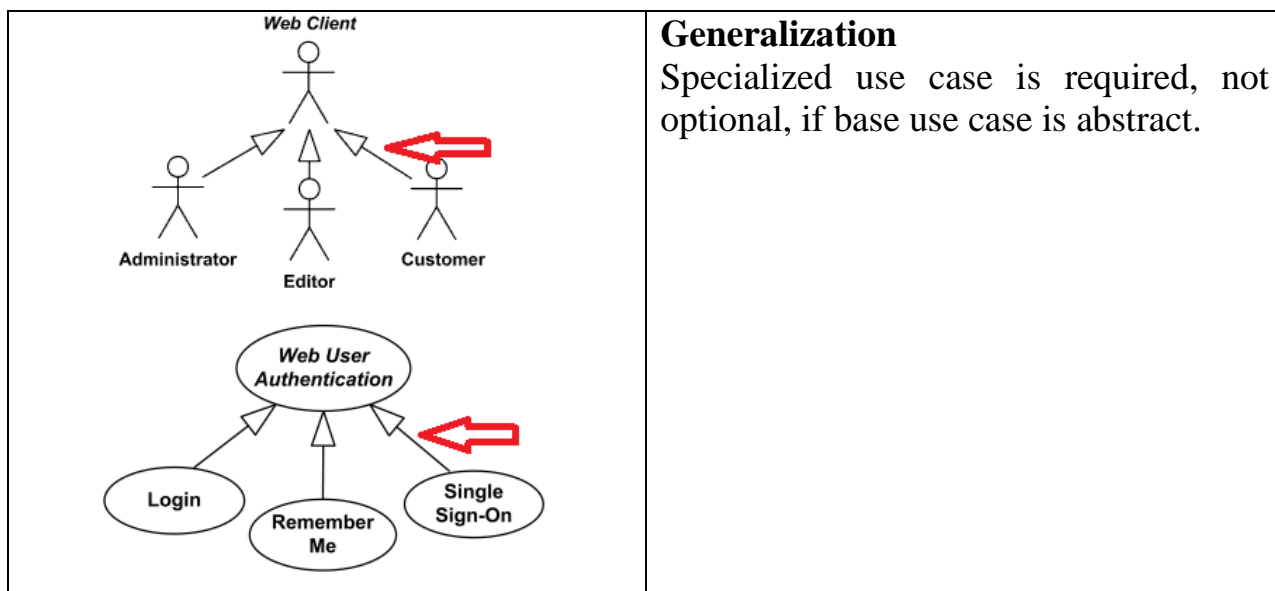
### 3. Develop the Use Case diagram.

Create Use Case diagram and set the name using: “File → New Diagram → New Use Case Diagram” menu; or click the “Use Case Diagram” menu in the Diagram Navigator and then select the “New Use Case Diagram”.

**Use case diagrams** are used to describe a set of actions (use cases) that some system should or can perform in collaboration with one or more external users of the system (actors). The Use Case diagram reflects functional requirements of software. All main elements of Use Case diagram are described in table 2.

Table 2 – Main elements of Use Case Diagram

	<b>Use cases</b> describe functionality provided by systems, and determine the requirements the systems pose on their environment.
	An <b>actor</b> is a person, organization, or external system that plays a role in one or more interactions with your system.
	<b>System boundary boxes (optional)</b> indicates the scope of system.
	An <b>association</b> between an actor and a use case indicates that the actor and the use case somehow interact or communicate with each other. An actor could be associated to one or several use cases.
	<b>Extend</b> Extending use case is optional, supplementary.
	<b>Include</b> Included use case required, not optional.



The free specialized software Visual Paradigm Community Edition is recommended to draw the Use Case diagram. The Use Case diagram for the given task is shown on Figure 1.5.

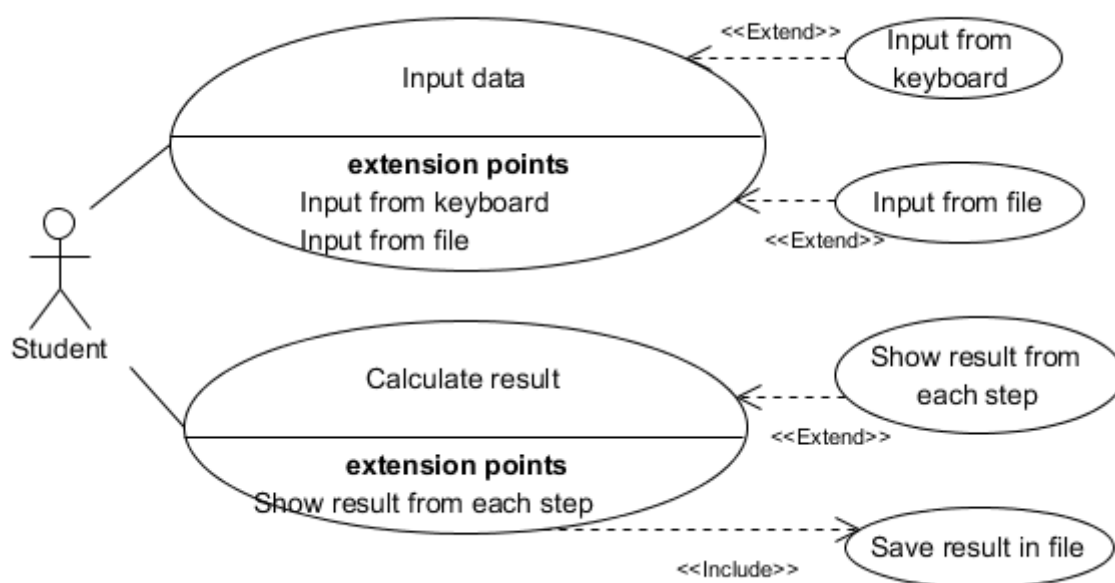


Figure 1.5 – Use Case Diagram for chosen task



#### **4. Prepare the report of the work**

Requirements for the report:

1. The report shall be drawn up according to the department standards.  
Template can be found by the following path:  
\\SELENA\Method\Paper work\STVUZ.dot
2. The report shall contain the results of implementation of all the tasks of the lab in accordance with the individual task.

## Lab № 2

### Goal: Learning basic principles of C++

Tasks:

1. Develop the code for the task from previous lab.
2. Prepare the report of the work

Progress of the lab.

#### 1. Develop the code for the task from previous lab.

According to the task from the previous lab you should develop computational algorithm in Visual Studio.

For example, you obtained the following task:

*The application has to calculate the values by the following formula for the range with a predefined step:*

$$y = \begin{cases} \sum_{i=2}^{n-2} (x-i)^2, & x < 7 \\ x + 7, & x \geq 7 \end{cases} \quad (2.1)$$

*Solution.*

First, to calculate the formula you should input the values of  $x$  and  $n$ . The following constraint: value of  $n$  should be greater than 4:

$$\sum_{i=2}^{n-2} (x-i)^2 = \sum_{i=2}^{4-2} (x-i)^2 = \sum_{i=2}^2 (x-i)^2 = (x-i)^2$$

The following are some examples:

$$n = 5; x = 6 \Rightarrow y = \sum_{i=2}^{5-2} (x-i)^2 = \sum_{i=2}^3 (x-i)^2 = (6-2)^2 + (6-3)^2 = 25$$

$$n = 5; x = 8 \Rightarrow y = x + 7 = 8 + 7 = 15$$

Create a new Visual Studio C++ Project. On the menu bar, choose File→New→Project (Figure 2.1.).

Author: Melnik K.V., PhD assistant

Department of software engineering and management information technologies, NTU “KhPI”  
Kharkiv, 2016

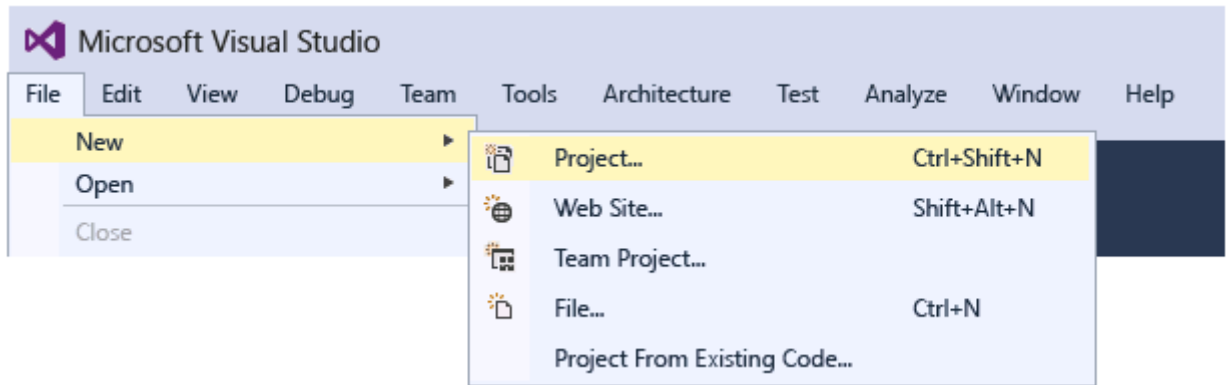


Figure 2.1 – Process of creating a blank project

In the Visual C++ category, choose the Win32 Console Application template, and then name the project (Figure 2.2).

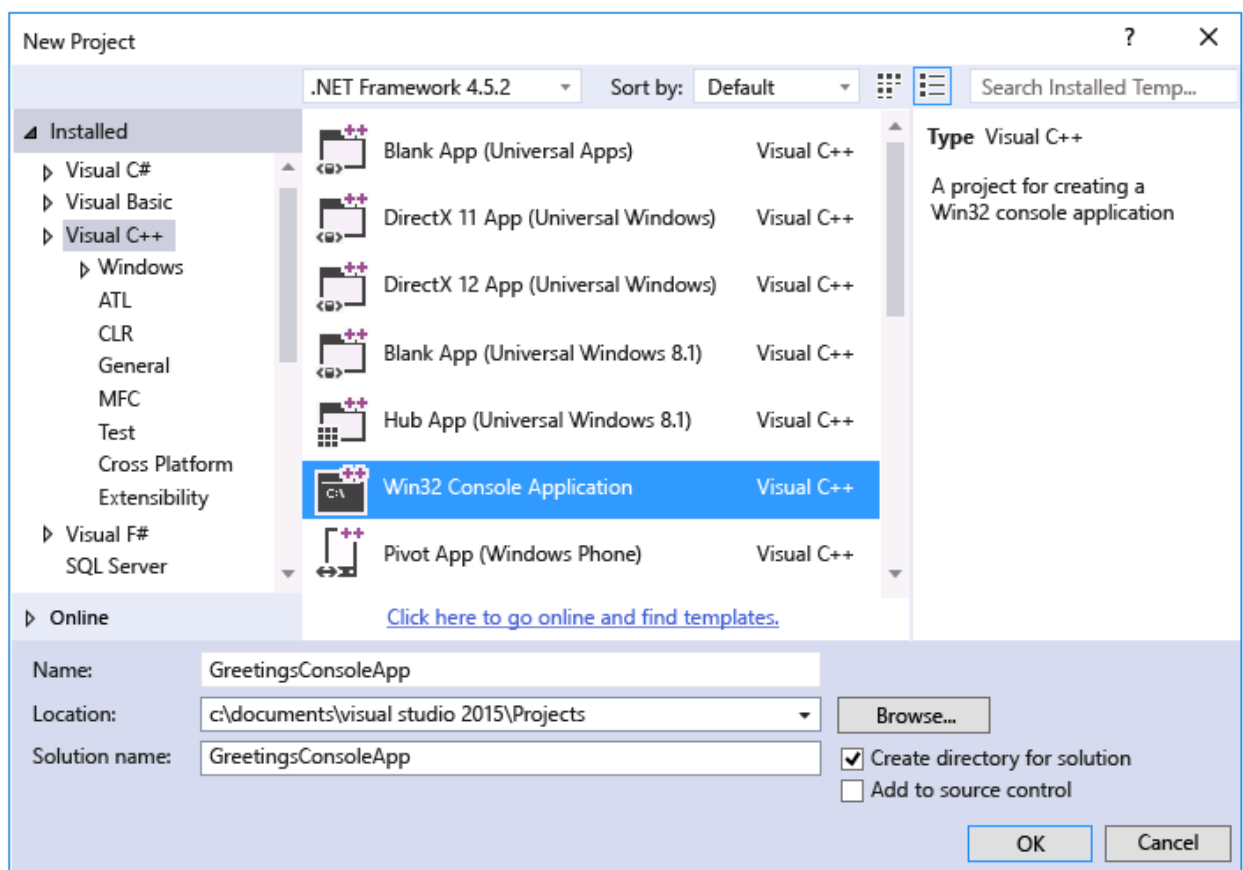


Figure 2.2 – Process of creating a blank project

When the Win32 Application Wizard appears, choose the Finish button. Implement the task from the previous lab.

The program for calculation of the function  $y$  is given on Figure 2.3. Samples calculated by the program can be found on Figure 2.4.

```

1  #include <iostream>
2  #include <math.h>
3  using namespace std;
4
5  int main()
6  {
7
8  int x,n;
9  double y=0;
10
11     cout << "Input n>=4, n=";
12     cin >> n;
13     cout << "Input x= " ;
14     cin >> x;
15
16     if (x<7)
17     {
18         for (int i=2; i<=n-2; i++)
19         {
20             y+=pow((x-i),2);
21         }
22     }
23     else
24     {
25         y=x+7;
26     }
27     cout << "x= " << x << " "; " << "y= " << y << endl;
28     return 0;
29 }

```

Figure 2.3 – Program for chosen task

```

Input n>=4, n=5
Input x= 6
x= 6; y= 25

```

```

Input n>=4, n=5
Input x= 8
x= 8; y= 15

```

Figure 2.4 – Result of execution the program

The next step is to modify the program. It should have the following possibilities:

Author: Melnik K.V., PhD assistant

Department of software engineering and management information technologies, NTU “KhPI”  
Kharkiv, 2016

- you should input predefined range (a,b) and step;
- program should calculate function  $y$  for each value  $x$  from range (a,b) (Figure 2.5);
- the results of each step should be displayed on the screen.

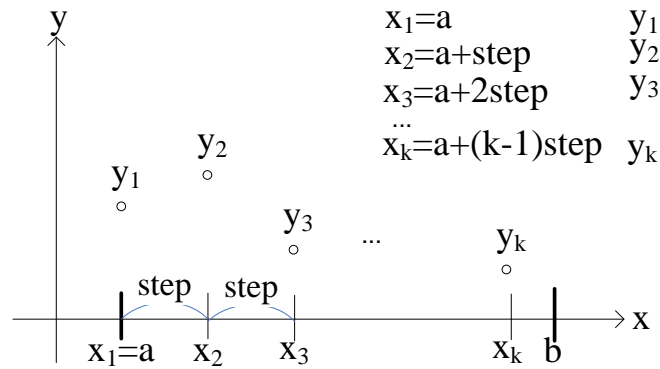


Figure 2.5 – Graphical representation of calculation

## 2. Prepare the report of the work

Requirements for the report:

1. The report shall be drawn up according to the department standards.  
Template can be found by the following path:  
\\SELENA\Method\Paper work\STVUZ.dot
2. The report shall contain the results of implementation of all the tasks of the lab in accordance with the individual task.

### Lab № 3

#### Goal: Learning basic principles of testing C++ code

##### Tasks:

1. Study principles of using functions in C++.
2. Study Exception Handling in C++.
3. Modify the code from previous lab according to 1 and 2 tasks.
4. Implement unit testing for developed program.
5. Prepare the report of the work

##### Progress of the lab.

#### 1. Study principles of using functions in C++.

You should modify the code from the previous lab by using functions in C++.

A **function** is a group of statements that is given a name, and which can be called from some point of the program. Functions allow to structure programs in segments of code to perform individual tasks. The main advantage is a function can actually be called multiple times within a program.

Depending on whether a function is predefined or created by programmer; there are two types of function:

1. **Library Function.** Programmer can use library function by invoking function directly; they don't need to write it themselves.
2. **User-defined Function.** A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).

The most common syntax to define a function is:

```
type name ( parameter1, parameter2, ...) { statements }
```

where:

- `type` is the type of the value returned by the function; if no value is returned to the calling function then, `void` should be used;

Author: Melnik K.V., PhD assistant

Department of software engineering and management information technologies, NTU “KhPI”  
Kharkiv, 2016

- `name` is the identifier by which the function can be called;
- `parameters` (as many as needed): The purpose of parameters is to allow passing arguments to the function from the location where it is called from;
- `statements` is the function's body.

## 2. Study Exception Handling in C++.

**An exception** is an error condition that prevents the program from continuing along its regular execution path. Certain operations, including object creation, file input/output, and function calls made from other modules, are all potential sources of exceptions even when your program is running correctly. Robust code anticipates and handles exceptions. The C++ language provides built-in support for throwing and catching exceptions. To implement exception handling in C++, you use `try`, `throw`, and `catch` expressions.

```
try
{
    // ...

    throw 1;
}
catch( ... )
{
    // Handle exception here.
}
```

Use a `try` block to enclose one or more statements that might throw an exception. A `throw` expression signals that an exceptional condition – often, an error – has occurred in a `try` block. The `throw` statement behaves much like `return`. You can use an object of any type as the operand of a `throw` expression. Typically, this object is used to communicate information about the error. To handle exceptions that may be thrown, implement one or more `catch` blocks immediately following a `try` block. Each `catch` block specifies the type of exception it can handle.

### 3. Modify the code from previous lab according to 1 and 2 tasks.

The function “Calculate” shows how to calculate value on definite range of expression (2.1) from previous lab (Figure 3.1). This expression has one specific condition: the value of  $n$  should be greater than 4. This constraint was developed in function “checkValidParams” (Figure 3.1).

If user enter incorrect value for  $n$  and  $x$ , for example, character or double value, then program should generate exception. These cases are reflected in the function “checkValidInput”.

```

1  #include "stdafx.h"
2  #include <iostream>
3  #include <math.h>
4  using namespace std;
5
6  void checkValidInput()
7  {
8      if (cin.fail())
9      {
10         throw "Incorrect input";
11     }
12 }
13
14 void checkValidParams(int n)
15 {
16     if (n < 4)
17     {
18         throw "Input correct data";
19     }
20 }
21
22 int calculate(int n, int x)
23 {
24     int y = 0;
25
26     if (x < 7)
27     {
28         for (int i = 2; i <= n - 2; i++)
29         {
30             y += pow((x - i), 2);
31         }
32     }
33     else
34     {
35         y = x + 7;
36     }
37
38     return y;
39 }
40

```

Figure 3.1 – Part of the code

The main function of developed code is on the Figure 3.2.



```

41  int main()
42  {
43      int x, n;
44
45      try
46      {
47          cout << "Input n>=4, n=";
48          cin >> n;
49          checkValidInput();
50          checkValidParams(n);
51
52          cout << "Input x=";
53          cin >> x;
54          checkValidInput();
55
56          cout << "x= " << x << "; " << "y= " << calculate(n, x) << endl;
57      }
58      catch (const char* ex)
59      {
60          cout << ex << endl;
61          return -1;
62      }
63      catch (...)
64      {
65          cout << "Unknown error" << endl;
66          return -2;
67      }
68  }

```

Figure 3.2 – The “Main” function

#### 4. Implement unit testing for developed program

By this moment you should have been developed a Win32 Console Application with all necessary functions for solving the equation. To test these functions, you have to create unit tests for each function separately. Sometimes it is necessary to create several unit tests for one function. It is done with the aim to fully test the behavior of each function with all possible input parameters.

To create unit tests for the existing project, select File → Add → New Project (Figure 3.3). Select Native Unit Test Project (Figure 3.4).

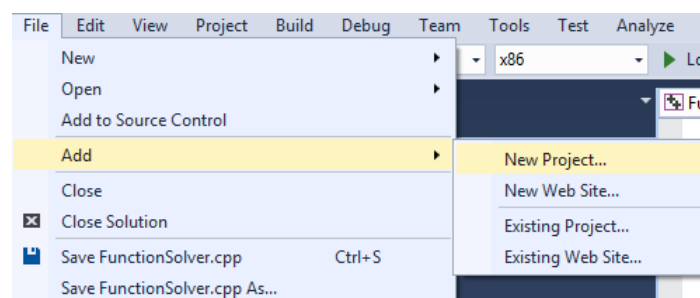


Figure 3.3 – The process of creating of unit test project

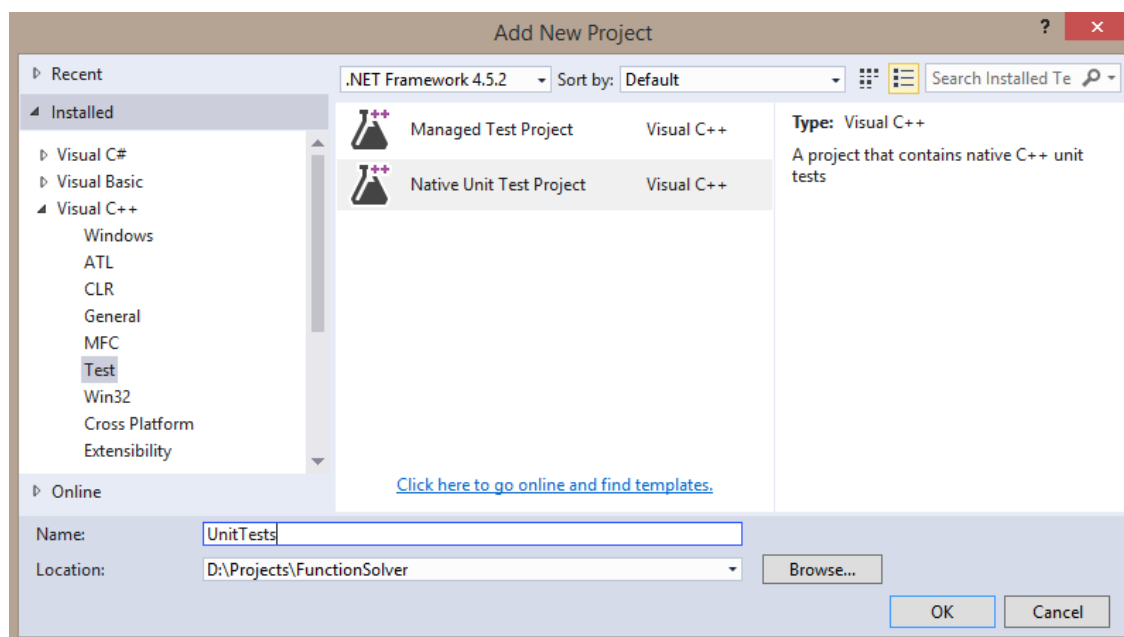


Figure 3.4 – The process of creating of unit test project

You can see unit test project in the Solution Explorer (Figure 3.5).

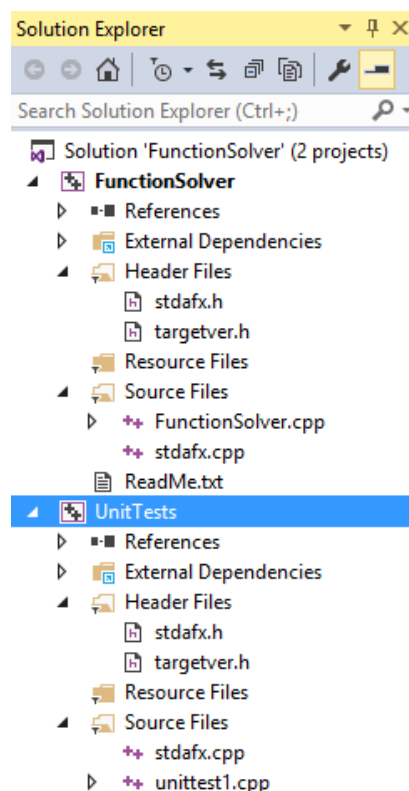


Figure 3.5 – Solution Explorer

Author: Melnik K.V., PhD assistant

Department of software engineering and management information technologies, NTU “KhPI”  
Kharkiv, 2016

Open the file `unittest1.cpp`. Unit tested will be placed in it. Right click on your source cpp file and choose “Copy Path” menu item (Figure 3.6). Paste this path to the file that will contain your unit tests (Figure 3.7).

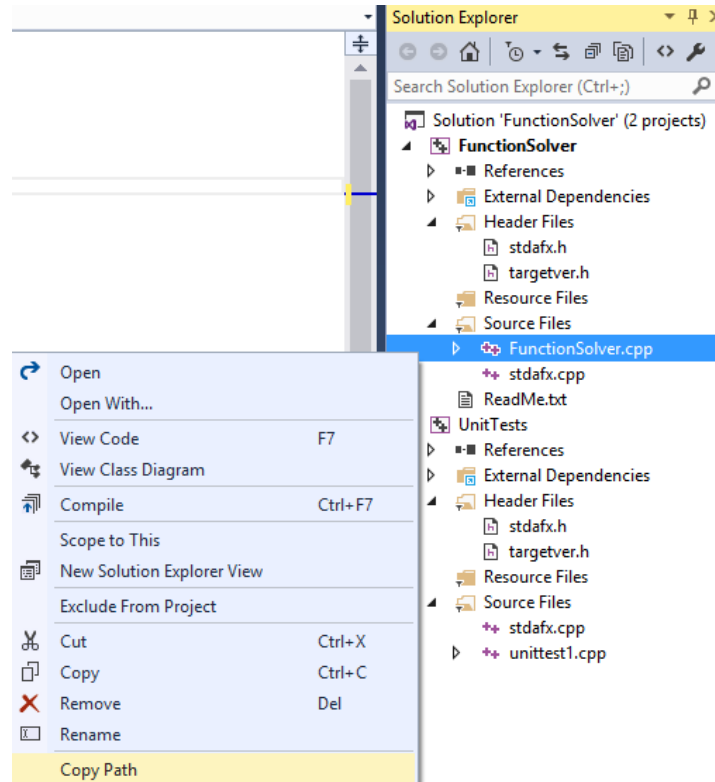


Figure 3.6 – Using source cpp file in test project (1)

```

1  #include "stdafx.h"
2  #include "CppUnitTest.h"
3  #include "D:\Projects\FunctionSolver\FunctionSolver\FunctionSolver.cpp"
4
5  using namespace Microsoft::VisualStudio::CppUnitTestFramework;
6
7  namespace UnitTests
8  {
9
10 }
```

Figure 3.7 – Using source cpp file in test project (2)

Create tests for your code. Each test is defined by using `TEST_METHOD(YourTestName) { ... }`.

You do not have to write a conventional function signature. The signature is created by the macro `TEST_METHOD`. The macro generates an instance function that returns void. It also generates a static function that returns information about the test method. This information allows the test explorer to find the method.

Test methods are grouped into classes by using `TEST_CLASS(YourClassName) { ... }`.

When the tests are run, an instance of each test class is created. Create the test methods names in the following manner:

`testedFunction_functionArguments_expectedResult.`

For testing purposes use several different methods on your choice from the Assert class. You can read about this class and its methods on MSDN <https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.testtools.unittesting.assert.aspx>.

The example of tests is shown on Figures 3.8-3.9.

```

1  #include "stdafx.h"
2  #include "CppUnitTest.h"
3  #include "D:\Projects\FunctionSolver\FunctionSolver\FunctionSolver.cpp"
4
5  using namespace Microsoft::VisualStudio::CppUnitTestFramework;
6
7  namespace UnitTests
8  {
9      TEST_CLASS(calculate_Tests)
10     {
11     public:
12         TEST_METHOD(calculate_get4and7_14returned)
13         {
14             int n = 4;
15             int x = 7;
16             int expected = 14;
17
18             int actual = calculate(n, x);
19
20             Assert::AreEqual(expected, actual);
21         }
22
23     public:
24         TEST_METHOD(calculate_get15and6_170returned)
25         {
26             int n = 15;
27             int x = 6;
28             int expected = 170;
29
30             int actual = calculate(n, x);
31
32             Assert::AreEqual(expected, actual);
33         }
34     };
35

```

Figure 3.8 – Unit tests for Calculate function

```

36 TEST_CLASS(checkValidParams_Tests)
37 {
38 public:
39     TEST_METHOD(checkValidParams_get10_exceptionNotThrown)
40     {
41         int n = 10;
42
43         try
44         {
45             checkValidParams(n);
46             Assert::IsTrue(true);
47         }
48         catch (...)
49         {
50             Assert::Fail();
51         }
52     }
53
54 public:
55     TEST_METHOD(checkValidParams_get3_exceptionThrown)
56     {
57         int n = 3;
58
59         try
60         {
61             checkValidParams(n);
62             Assert::Fail();
63         }
64         catch (...)
65         {
66             Assert::IsTrue(true);
67         }
68     }
69 };
70

```

Figure 3.9 – Unit tests for checkValidParams function

When test methods are developed, you can execute them. To do that, you have to build solution (Build → Build Solution / Ctrl + Shift + B), open Test Explorer if it wasn't opened automatically (Test → Windows → Test Explorer) and run all tests. If everything go smoothly, you will see that all tests passed in the summary sub-window (Figure 3.10).

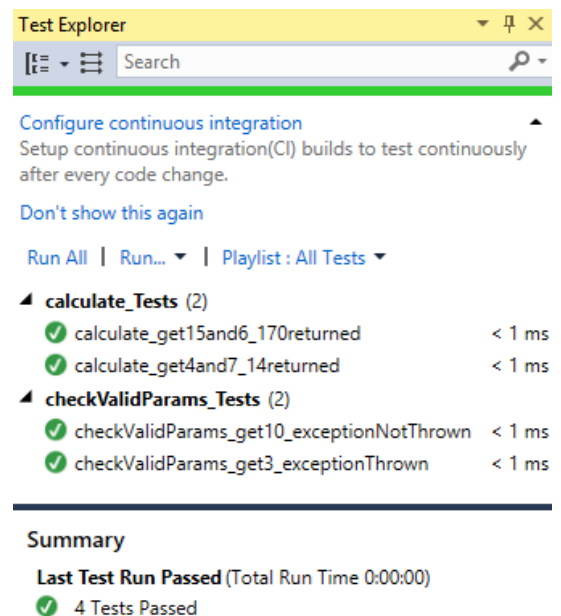


Figure 3.10 – Successful unit tests

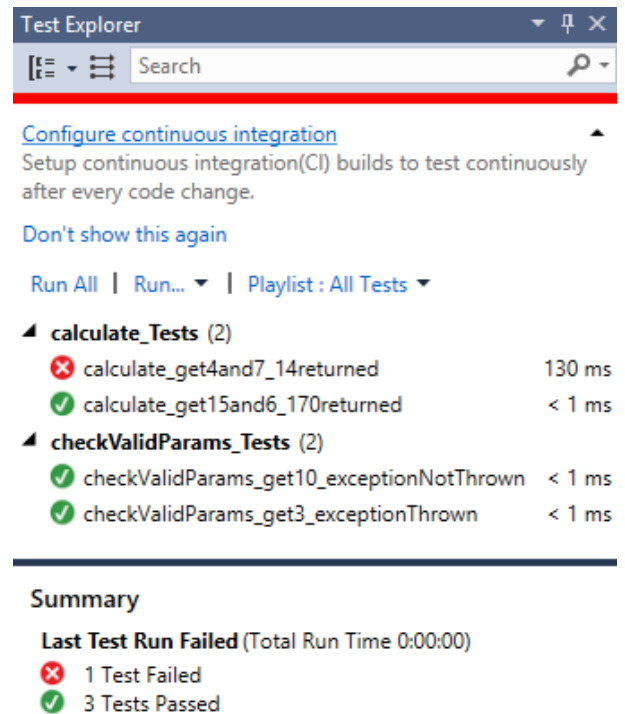
Also you should make some changes in unit tests to get them unpassed. For example, change the expected result to the wrong value and run the test again (Figure 3.11 a). In the Test Explorer you will see which tests weren't passed (Figure 3.11 b).

```
TEST_METHOD(calculate_get4and7_14returned)
{
    int n = 4;
    int x = 7;
    int expected = 100000000;

    int actual = calculate(n, x);

    Assert::AreEqual(expected, actual);
}
```

(a)



(b)

Figure 3.11 – Unit test with error

## 5. Prepare the report of the work

Requirements for the report:

1. The report shall be drawn up according to the department standards.  
Template can be found by the following path:  
\\SELENA\Method\Paper work\STVUZ.dot
2. The report shall contain the results of implementation of all the tasks of the lab in accordance with the individual task.

Author: Melnik K.V., PhD assistant

Department of software engineering and management information technologies, NTU “KhPI”  
Kharkiv, 2016

## **Lab № 4**

**Goal: Plan system activities. To obtain basic experience and knowledge to work with project tracking system**

Progress of the lab.

### **1. Register on the free xp-dev.com account**

In order to register you should have an email account (it is also possible to provide a free email account). In order to register go to <https://xp-dev.com/register> and create new free account Free account provides you with 10Mb of storage for the repository

### **2. Create new project**

Choose "Create new project" and give a name, abbreviation and description for the project Issue. Tracking Type should be XPDev. It is necessary to create 1 xpdev account for 1 team. Create new project Inside XpDev

### **3. Add new user to project.**

Choose the username and password to send invitation. New user can log in with the credentials sent to his/her email account. Choose the role of user (examine all roles first ). The administrator or writer are acceptable.

### **4. Create new Iteration.**

The modern software development approaches plan the project activities by stages (iteration). Every iteration should implement several software or system functionalities (user stories). So every laboratory start with new iteration that can have several user stories.

Go to Project tracking and choose "Create new iteration".

Name: any of one that describe setup process

Start End- today

### **5. Add new user stories and add task to implementation of user story**

### **6. Assign tasks to users of you team**

### **7. Create svn repository**

Author: Melnik K.V., PhD assistant

Department of software engineering and management information technologies, NTU "KhPI" Kharkiv, 2016

Go to Repository tab

Choose create new repository

Set the name of repository

Set Create Initial Directories switch (it will create additional directories to handle multiple version of project to handle customer requirements changes)

### **8. Test accessibility via svn.exe (can get from the instructor)**

Create empty directory

Use command `svn checkout <url> <path to directory> --username=<xpedv's username>` to make

initial checkout (check for the all team members)

Create test text file on the trunk directory

Use command `svn add <path to directory>` to add new file

Use command `svn commit -m 'message'` to put data to repository

Use command `svn update <path to directory>` to update data on local checkout

### **9. Prepare the report of the work**

Requirements for the report:

1. The report shall be drawn up according to the department standards.  
Template can be found by the following path:  
\\SELENA\Method\Paper work\STVUZ.dot
2. The report shall contain the results of implementation of all the tasks of the lab in accordance with the individual task.