**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»**

**ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ  
КАФЕДРА СИСТЕМНОГО ПРОЕКТУВАННЯ**

Лабораторна робота №4

**Модульне тестуваня (Unit-тести) та рефакторинг**

Виконав:

Студент ІV курсу

Групи ДА-21

Сарапулов Віктор

Київ – 2015

**Мета роботи:** оволодіти навичками створення програмного забезпечення за метолологією  
TDD та ознайомитися з процедурами рефакторинга.

**Короткі теоретичні відомості**

Розробка через тестування (англ. тест-Driven Development, TDD) - техніка розробки  
програмного забезпечення, яка грунтується на повторенні дуже коротких циклів розробки:  
спочатку пишеться тест, що покриває бажану зміну, потім пишеться код, який дозволить  
пройти тест, і під кінець проводиться рефакторинг нового коду до відповідних стандартів.  
Рефакторинг (англ. refactoring) - процес зміни внутрішньої структури програми, що не  
зачіпає її зовнішньої поведінки і має на меті полегшити розуміння її роботи. В основі  
рефакторинга лежить послідовність невеликих еквівалентних (тобто таких, що зберігають  
поведінку) перетворень. Оскільки кожне перетворення маленьке, програмісту легше  
простежити за його правильністю, і в той же час вся послідовність може привести до істотної  
перебудови програми і поліпшенню її узгодженості і чіткості.  
Модульне тестування, або юніт-тестування (англ. unit testing) - процес у програмуванні, що  
дозволяє перевірити на коректність окремі модулі вихідного коду програми. Ідея полягає в  
тому, щоб писати тести для кожної нетривіальної функції або методу. Це дозволяє досить  
швидко перевірити, чи не призвело чергову зміну коду до регресії, тобто до появи помилок у  
вже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок.

**Завдання**  
1. Розробити методику випробувань з використанням ISO/IEC/IEEE 29119.  
2. Розробити код програми архітектурної моделі. Використовувати Test Driven Development.  
3. Провести рефакторинг коду програми, щоб задовольнити вимоги технічного завдання.

**Зміст звіту**  
1. Мета роботи.  
2. Завдання роботи.  
3. Оформлення результатів роботи:  
3.1. Опис юніт тестів. Опис методити випробувань за ISO або IEEE стандартом.  
3.2. Опис застосування TestDrivenDevelopment для конкретної реалізації класів програмної  
моделі.  
3.3. Опис процедур рефакторинга для коду програми для конкретних випадків поліпшення  
API і архітектури класів.  
3.4. Опис методики випробувань з використанням ISO/IEC/IEEE 29119 або IEEE\_829

**Виконання роботи**

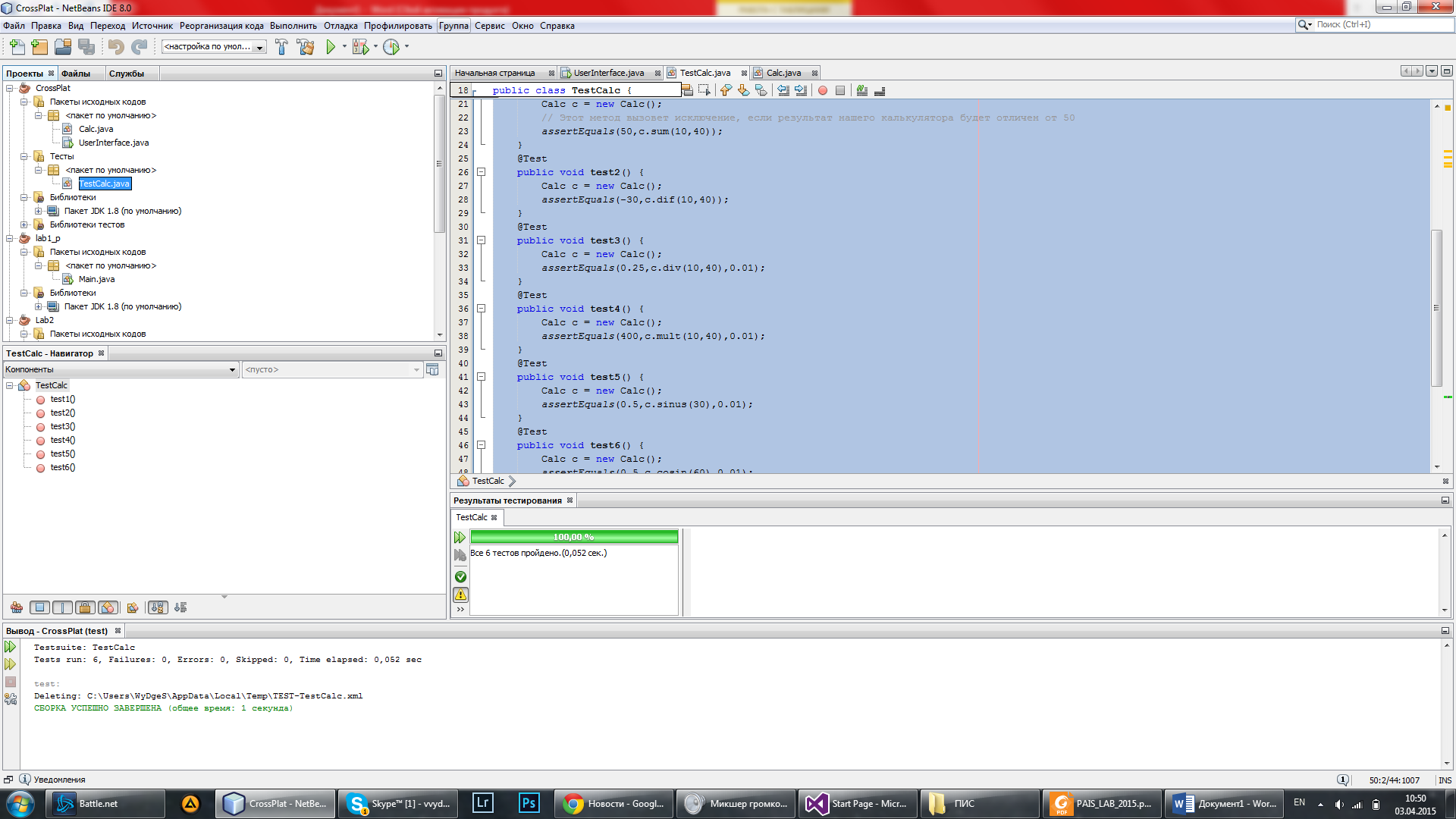
1. **Написання тесту до класу Calc.java**

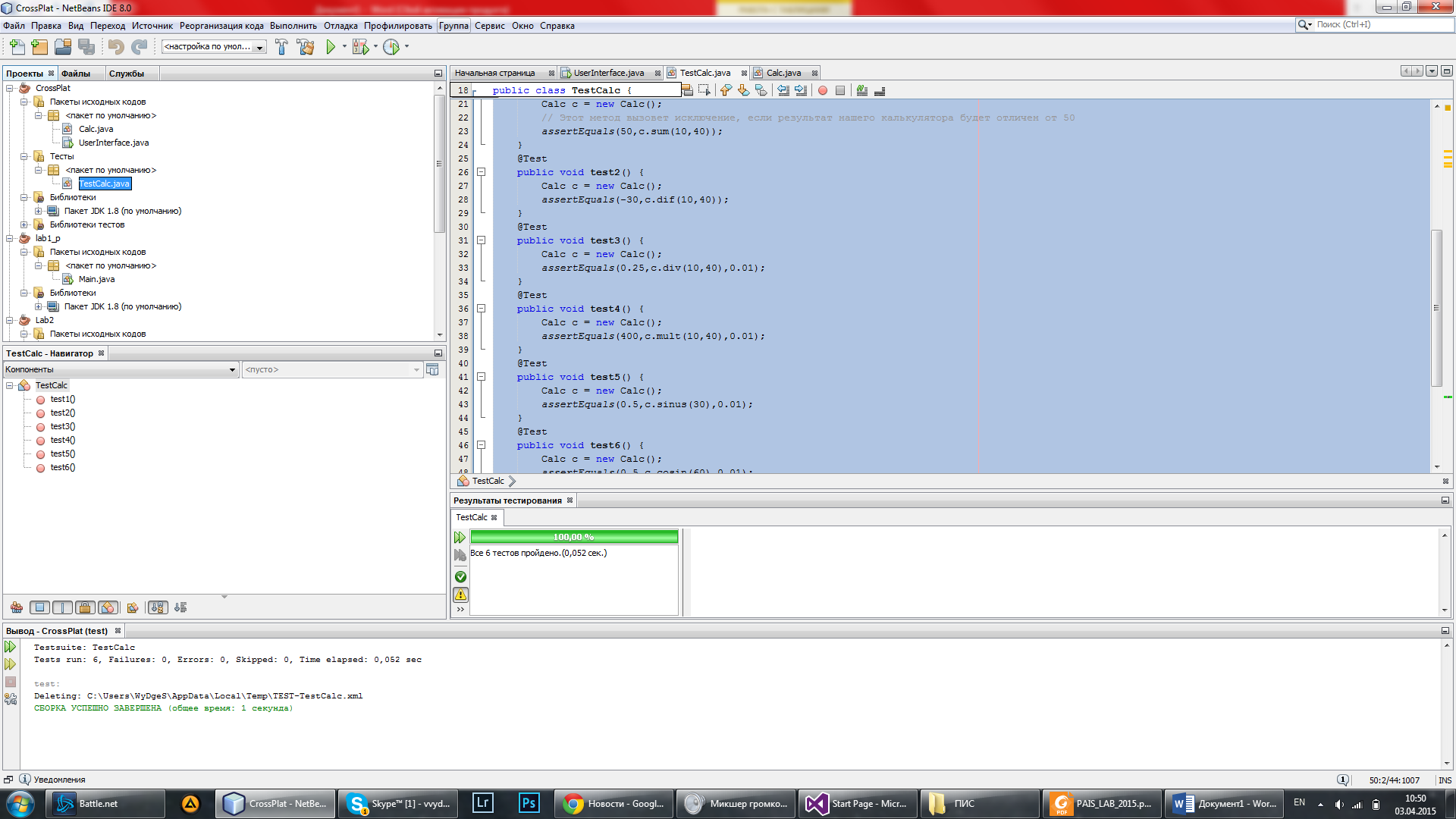
|  |
| --- |
| import org.junit.After;  import org.junit.AfterClass;  import org.junit.Before;  import org.junit.BeforeClass;  import org.junit.Test;  import static org.junit.Assert.\*;  public class TestCalc {  @Test  public void test1() {  Calc c = new Calc();  // Этот метод вызовет исключение, если результат нашего калькулятора будет отличен от 50  assertEquals(50,c.sum(10,40));  }  @Test  public void test2() {  Calc c = new Calc();  assertEquals(-30,c. sum (10,-40));  }  @Test  public void test3() {  Calc c = new Calc();  assertEquals(167,c. sum (91,76);  }  @Test  public void test4() {  Calc c = new Calc();  assertEquals(52546,c. sum (20000,32546));  }  @Test  public void test5() {  Calc c = new Calc();  assertEquals(167432,c. sum (46789,120643));  }  @Test  public void test6() {  Calc c = new Calc();  assertEquals(245,c.sum(161,84));  }  } |

1. **Написання класу**

|  |
| --- |
| public class Calc{  public int sum(int x, int y){  return x+y;  }  } |

1. **Результати тестування**





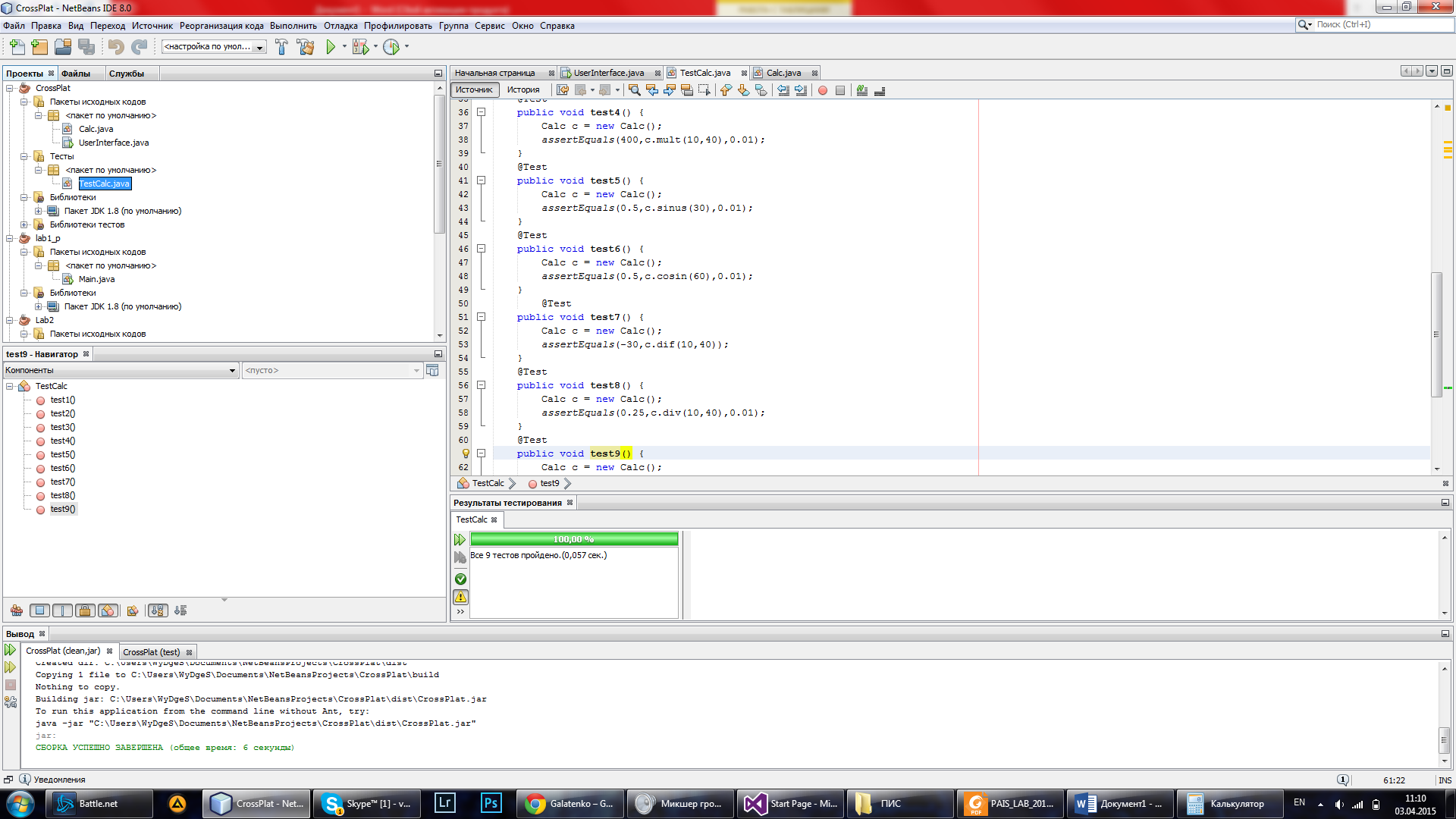
1. **Ускладнення тесту**

|  |
| --- |
| @Test  public void test7() {  Calc c = new Calc();  assertEquals(-30,c.div(10,40));  }  @Test  public void test8() {  Calc c = new Calc();  assertEquals(0.25,c.div(10,40),0.01);  }  @Test  public void test9() {  Calc c = new Calc();  assertEquals(400,c.mult(10,40),0.01);  } |

1. **Ускладнення класу**

|  |
| --- |
| public class Calc{  public int sum(int x, int y){  return x+y;  }  }  public int mult(int x, int y){  return x\*y;  }  public double div(int x, int y){  return (double)x/(double)y;  } |

1. **Результати**



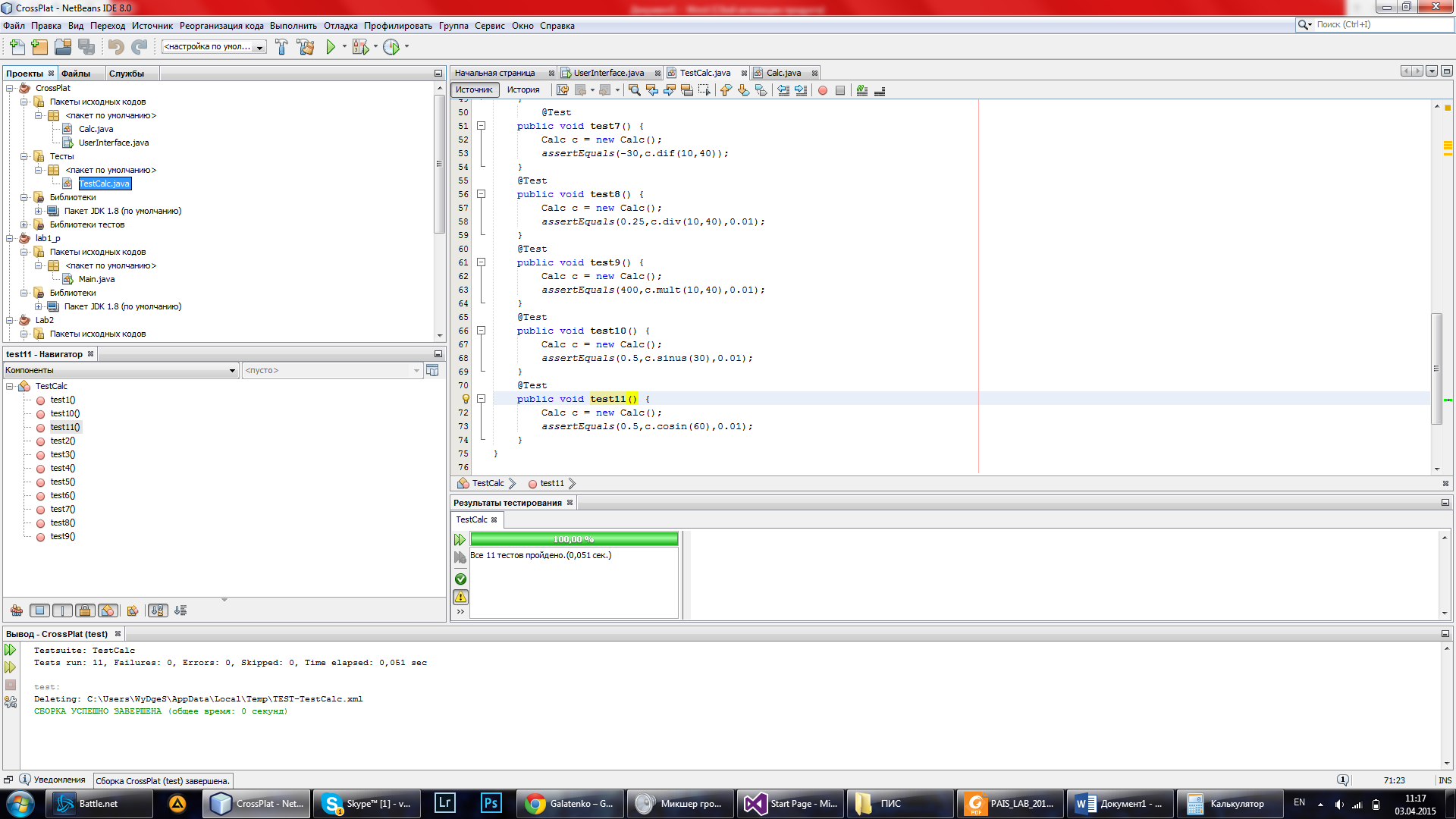
1. **Ускладення тесту**

|  |
| --- |
| @Test  public void test5() {  Calc c = new Calc();  assertEquals(0.5,c.sinus(30),0.01);  }  @Test  public void test6() {  Calc c = new Calc();  assertEquals(0.5,c.cosin(60),0.01);  } |

1. **Ускладнення**

|  |
| --- |
| public class Calc{  public int sum(int x, int y){  return x+y;  }    public int dif(int x, int y){  return x-y;  }    public int mult(int x, int y){  return x\*y;  }  public double div(int x, int y){  return (double)x/(double)y;  }  public double cosin(int x){  double radians=Math.toRadians(x);  return Math.cos(radians);  }    public double sinus(int x){  double radians=Math.toRadians(x);  return Math.sin(radians);  }  } |

1. **Результати**



**Висновок:** В ході даної лабораторної роботи був отриманий досвід у створення програмного забезпечення за методологією TDD. Освоєно тестування технологією JUnit.