Relatorio DEVELOPER

Alan Rafael Teixeira Soledar
Fernando Gomes
Maria Eduarda Ruhe
Leonardo de Souza Augusto

Visão geral do projeto código

Não tivemos tempo de terminar todas as funcionalidades, encontramos diversos problemas inesperados na lógica do programa, e não conseguimos criar muitas classes de teste para a lógica da domain pois foi difícil de pensar em como fazer testes tanto sem ter as funções como com funções tão complexas que para cada teste precisava de vários dados. A classe User teve testes uteis.

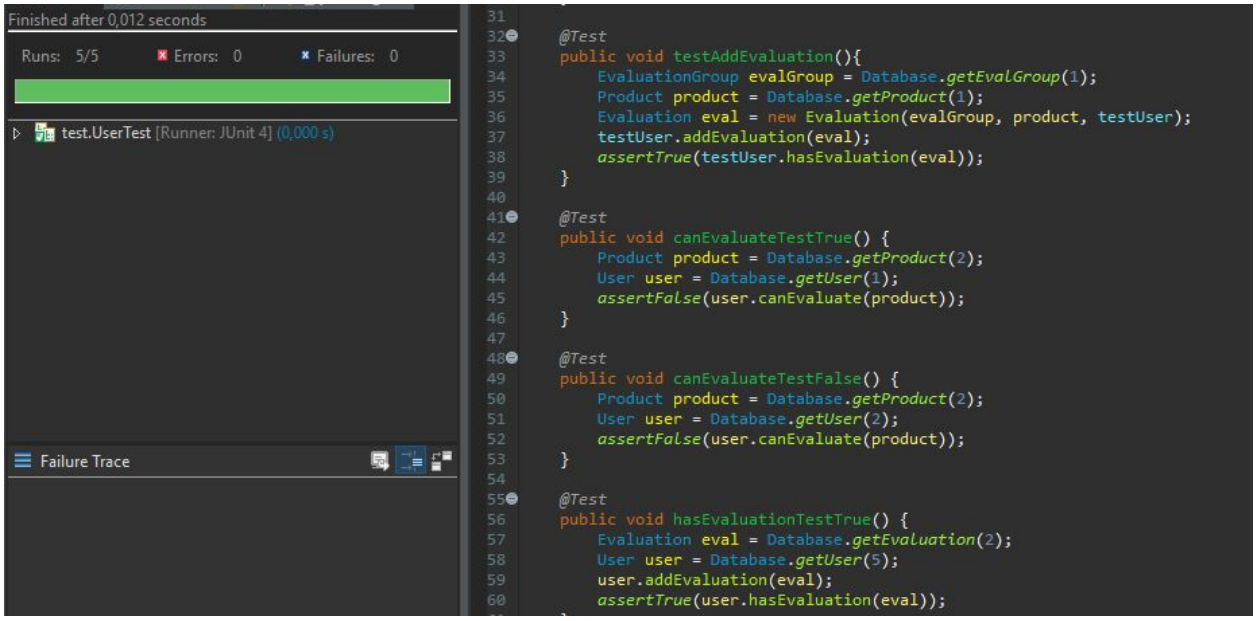
Apesar disso as classes Command, que são a implementação da interface e os chamamentos dos métodos da lógica principal do programa, funcionaram bem em conjunto com as classes da domain.

Visão geral das modificações no projeto original

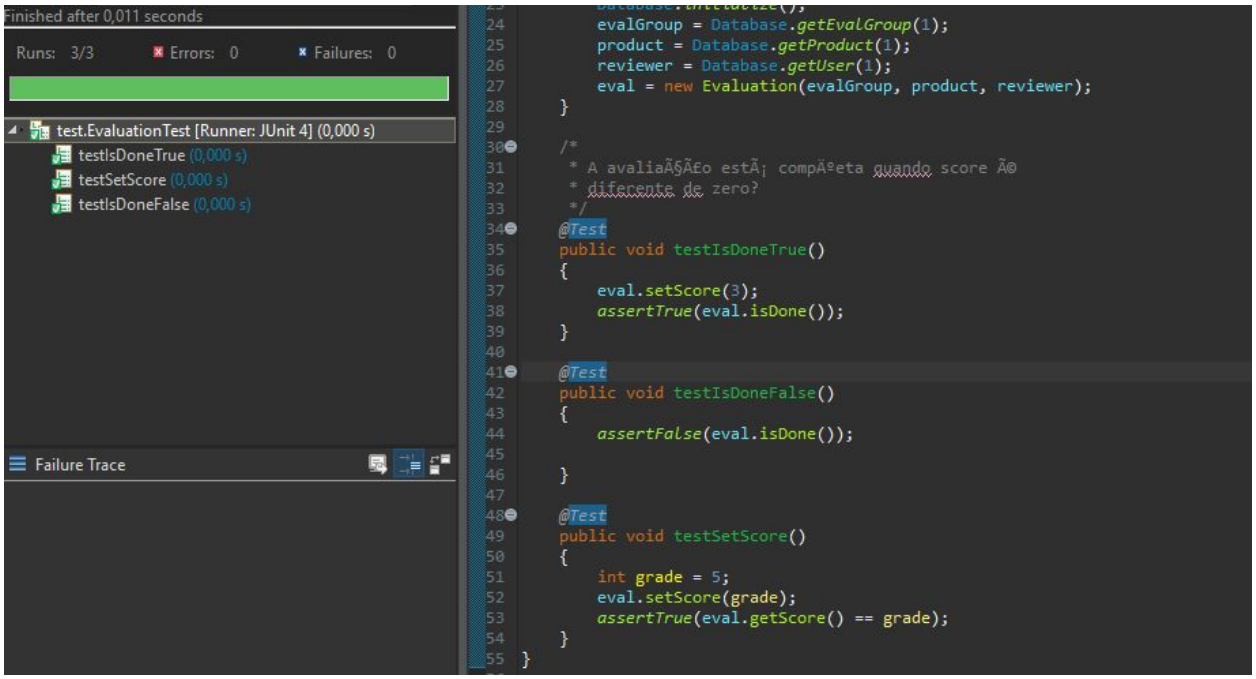
Em geral o projeto foi mantido como no UML, pois não ficou tão diferente do que tínhamos imaginado. Tínhamos imaginado usar uma classe para o report que acabamos não usando mas fora isso todas os métodos foram aproveitados ao máximo, alguns poucos ficaram sem implementação ou foram implementados e acabamos não utilizando.

Visão geral dos testes unitários realizados

Não conseguimos pensar em como fazer testes para a classe EvaluationGroup pois não imaginamos como testar função no estilo da allocate que não tem retorno é muito complexa. A classe User e Evaluation eram simples o suficiente para fazer testes de assertTrue e assertFalse




```
31
32      @Test
33      public void testAddEvaluation(){
34          EvaluationGroup evalGroup = Database.getEvalGroup(1);
35          Product product = Database.getProduct(1);
36          Evaluation eval = new Evaluation(evalGroup, product, testUser);
37          testUser.addEvaluation(eval);
38          assertTrue(testUser.hasEvaluation(eval));
39      }
40
41      @Test
42      public void canEvaluateTestTrue() {
43          Product product = Database.getProduct(2);
44          User user = Database.getUser(1);
45          assertFalse(user.canEvaluate(product));
46      }
47
48      @Test
49      public void canEvaluateTestFalse() {
50          Product product = Database.getProduct(2);
51          User user = Database.getUser(2);
52          assertFalse(user.canEvaluate(product));
53      }
54
55      @Test
56      public void hasEvaluationTestTrue() {
57          Evaluation eval = Database.getEvaluation(2);
58          User user = Database.getUser(5);
59          user.addEvaluation(eval);
60          assertTrue(user.hasEvaluation(eval));
```




```
24          evalGroup = Database.getEvalGroup(1);
25          product = Database.getProduct(1);
26          reviewer = Database.getUser(1);
27          eval = new Evaluation(evalGroup, product, reviewer);
28      }
29
30      /*
31       * A avaliaÃ§Ã£o estÃ¡ compÂºeta quando score Ã©
32       * difecente de zero?
33       */
34      @Test
35      public void testIsDoneTrue()
36      {
37          eval.setScore(3);
38          assertTrue(eval.isDone());
39      }
40
41      @Test
42      public void testIsDoneFalse()
43      {
44          assertFalse(eval.isDone());
45
46      }
47
48      @Test
49      public void testSetScore()
50      {
51          int grade = 5;
52          eval.setScore(grade);
53          assertTrue(eval.getScore() == grade);
54      }
55  }
```