Software Testing

CA5 Report

Pit Test Coverage Report

Package Summary

domain

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
2	98%	48/49	97%	28/29	97%	28/29

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Engine.java	98%	39/40	95%	20/21	95%	20/21
Order.java	100%	9/9	100%	8/8	100%	8/8

تعداد mutant های ساخته شده

تعداد mutant های کشته شده برای هر دو کلاس را بررسی می‌کنیم:

Engine.java

```
1   package domain;
2
3   import java.util.ArrayList;
4
5   public class Engine {
6
7       ArrayList<Order> orderHistory;
8
9       public Engine() {
10          orderHistory = new ArrayList<>();
11      }
12
13      int getAverageOrderQuantityByCustomer(int customer) {
14          var sum = 0;
15          var count = 0;
16
17          for (Order oldOrder : orderHistory) {
18  1           if (oldOrder.customer == customer) {
19  1               sum += oldOrder.quantity;
20  1               count++;
21              }
22          }
23
24  1       if (orderHistory.size() == 0) {
25              return 0;
26          }
27
28  2       return sum / count;
29      }
30
31      int getQuantityPatternByPrice(int price) {
32  1       if (orderHistory.size() == 0) {
33              return 0;
34          }
35
36          var diff = 0;
37          var previous = orderHistory.get(0);
38
39          for (Order currentOrder : orderHistory) {
40  1           if (currentOrder.id == previous.id) {
41                  continue;
42              }
43
44  1           if (currentOrder.price != price) {
45                  continue;
```

```
44 1             if (currentOrder.price != price) {
45                   continue;
46               }
47
48 1             if (diff == 0) {
49 1                 diff = currentOrder.quantity - previous.quantity;
50                   previous = currentOrder;
51 2             } else if (diff != currentOrder.quantity - previous.quantity) {
52                   return 0;
53               }
54           }
55
56 1         return diff;
57       }
58
59       int getCustomerFraudulentQuantity(Order order) {
60
61           var averageOrderQuantity = getAverageOrderQuantityByCustomer(order.customer);
62
63 2         if (order.quantity > averageOrderQuantity) {
64 2             return order.quantity - averageOrderQuantity;
65           }
66
67           return 0;
68       }
69
70       public int addOrderAndGetFraudulentQuantity(Order order) {
71 1         if (orderHistory.contains(order)) {
72               return 0;
73           }
74
75           var quantity = getCustomerFraudulentQuantity(order);
76 1         if (quantity == 0) {
77               quantity = getQuantityPatternByPrice(order.price);
78           }
79
80           orderHistory.add(order);
81 1         return quantity;
82       }
83   }
```

Mutations

18	1. negated conditional → KILLED
19	1. Replaced integer addition with subtraction → KILLED
20	1. Changed increment from 1 to -1 → KILLED
24	1. negated conditional → KILLED
28	1. replaced int return with 0 for domain/Engine::getAverageOrderQuantityByCustomer → KILLED 2. Replaced integer division with multiplication → KILLED
32	1. negated conditional → KILLED
40	1. negated conditional → KILLED
44	1. negated conditional → KILLED
48	1. negated conditional → KILLED
49	1. Replaced integer subtraction with addition → KILLED
51	1. negated conditional → KILLED 2. Replaced integer subtraction with addition → KILLED
56	1. replaced int return with 0 for domain/Engine::getQuantityPatternByPrice → KILLED
63	1. changed conditional boundary → SURVIVED 2. negated conditional → KILLED
64	1. replaced int return with 0 for domain/Engine::getCustomerFraudulentQuantity → KILLED 2. Replaced integer subtraction with addition → KILLED
71	1. negated conditional → KILLED
76	1. negated conditional → KILLED
81	1. replaced int return with 0 for domain/Engine::addOrderAndGetFraudulentQuantity → KILLED

برای این کلاس ۱۸ mutant ساخته شد و تنها یکی از آن‌ها survive کرد.

Order.java

```
1   package domain;
2
3   import lombok.Getter;
4   import lombok.Setter;
5
6   @Getter
7   @Setter
8   public class Order {
9  1    int id;
10 1    int customer;
11 1    int price;
12 1    int quantity;
13
14      @Override
15      public boolean equals(Object obj) {
16 1        if (obj instanceof Order order) {
17 2            return id == order.id;
18          }
19 1        return false;
20      }
21  }
```

Mutations

```
9   1. replaced int return with 0 for domain/Order::getId → KILLED
10  1. replaced int return with 0 for domain/Order::getCustomer → KILLED
11  1. replaced int return with 0 for domain/Order::getPrice → KILLED
12  1. replaced int return with 0 for domain/Order::getQuantity → KILLED
16  1. negated conditional → KILLED
17  1. negated conditional → KILLED
    2. replaced boolean return with true for domain/Order::equals → KILLED
19  1. replaced boolean return with true for domain/Order::equals → KILLED
```

برای این کلاس ۸ mutant ساخته شد که همه‌ی آن‌ها KILL شدند.

تاثیر coverage mutation بالا در میزان خطر refactoring

افزایش Coverage Mutation به معنای پوشش گسترده‌تر تست‌های Mutation است و میتواند به کاهش خطرات و کاهش احتمال وجود باگ‌ها و خطاها کمک کند. اگر تست‌های Mutation نشان دهنده خطاهای احتمالی را بشناسند، افزایش پوشش این تستها به تضمین ایمنی کد کمک خواهد کرد. تست‌های Mutation همچنین می توانند خطاهای مخفی و نقاط ضعف کد را شناسایی کنند که تست‌های معمولی این امکان را ندارند. اگر تست‌های Mutation بیشتري پوشش داده شود، احتمالاً میزان اطمینان از اصلاحات انجام شده در فرآیند Refactoring افزایش خواهد یافت. افزایش پوشش تست‌های Mutation ممکن است با هزینه و زمان بیشتری همراه باشد. بسته به ماهیت پروژه، این افزایش هزینه ممکن است به میزان قابل توجهی باشد. تست‌های Mutation بهترین نتایج را زمانی خواهند داشت که به عنوان یک بخش از استراتژی کلی تست‌گذاری در نظر گرفته شوند. تست‌های Mutation باید با تست‌های واحد و تست‌های سیستم ترکیب شوند تا تضمین شود که همه‌ی جنبه‌های کد مورد بررسی و آزمایش قرار گرفته‌اند.

WORKFLOW AND PIPELINE