# Software Engineering Methods: Assignment 3

Group 04a: Bogdan Iordache, Kyongsu Kim, Matyáš Kollert, Anna Lantink,
Kristóf Lévay, Javier Páez Franco

January 25th, 2022

## 1 Automated Mutation Testing

### 1.1 Topping

The Topping class had a mutation coverage of 63% as you can see in Figure 1. After improving the test suite, the mutation coverage increased to 88% (see Figure 2). The two related commits are: b24a52b4, and 7b3171c9.

### 1.2 Allergen

The Allergen class had a mutation coverage of 69% as you can see in Figure 3. After improving the test suite, the mutation coverage increased to 100% (see Figure 4). The related commit is: 9cbe23c1

### 1.3 StoreService

The original metrics are shown in Figure 5, with 14% mutation score. The final metrics can be found in Figure 6, with 100% mutation score. The related commit is: d1db7b62.

### 1.4 OrderUtils

The original metrics are shown in Figure 7, with 57% mutation score. The final metrics can be found in Figure 8, with 93% mutation score. The related commit is: 5bcb5cf1 .

## 2 Manual Mutation Testing

We chose to do manual mutation testing on the Order domain, meaning all functionality relating to making, sending, modifying, and cancelling an Order. This excludes any coupon or store functionality since those are in different domains, despite being in the same microservice.

### 2.1 getOrder(...) in OrderService Class

The OrderService class is critical because it handles and delegates all the core business logic for the Order domain. The getOrder method in the OrderService class queries the repository for the order with a given orderId, and if the order doesn't exist, it throws an error (Figure 9). The exception it throws communicates to the user what went wrong with their query. Without this error checking, the user would get no information as to why their request went wrong.

The mutation that was introduced was removing the conditional that checks if an Order is empty (Figure 10). Originally, this mutation was not killed, so tests were added in the GetOrderTests method to ensure that this conditional was checked. The mutant was killed in commit a8b84a82.

### 2.2 checkStatus(...) in OrderHelperService Class

The OrderHelperService is critical because its main purpose is to validate input data for the Order domain, and to communicate with other microservices. The checkStatus method in OrderHelperService checks if an order has already been paid or cancelled, in which case it throws the corresponding error (Figure 11). This method is important because it is used before modifying an order to check if a modification is indeed possible, since it makes no sense to modify an order that is already paid or cancelled.

The mutation that was introduced was to remove all of this method's functionality (Figure 12). This mutation was not killed by the tests, presumably because this method was always mocked during tests. 6 tests were introduced in the ModifyOrderTests class to kill the mutant. The mutant was killed in commit 64c21db4.

### 2.3 removeProductsFromOrder(...) in OrderController Class

Since every request concerning orders is going through the OrderController, it is crucial. This is the class that defines how the HTTP requests should be handled when contacting the REST API for orders, in this specific case, when the user wants to remove a product from their order. All requests to the "removeProductsFromOrder" endpoint are handled through the method of the same name. (Figure 13)

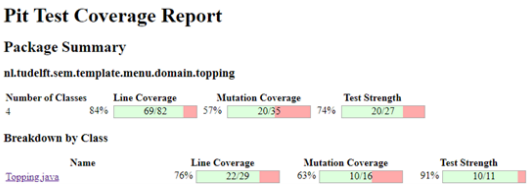
The method checks if the user is indeed the one who created the order that they try to modify. If not, it throws an error. There is a condition in the beginning of the method which checks if the request contains a list of product IDs to delete. The injected bug is that in `request.getProducts() != null`, the operator was changed to "==". (Figure 14) This mutant was killed with commit 7f73f34a.

### 2.4 removeToppingsFromProduct(...) in ModifyOrderService Class

The ModifyOrderService class is critical since it executes the removal of products or toppings from an order. The removeToppingsFromProduct method in this class removes multiple toppings from the product, which is a critical part of modifying an order. This method throws InvalidToppingsException if the provided toppings are null, which is a vital boundary check that ensures input data is valid (Figure 15). The exception ensures that the user knows what went wrong with the query.

The mutation introduced in this case was replacing the process of checking whether the toppings are null to checking whether the toppings are not null (Figure 16). Originally, this mutation was not killed, so tests was added into testRemoveNullTopping method in ModifyOrderTest class. The mutant is killed in commit a1449fe6

## 3 Appendix



**Pit Test Coverage Report**

**Package Summary**

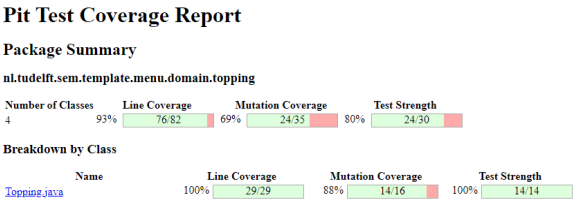
**nl.tudelft.sem.template.menu.domain.topping**

| Number of Classes | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| 4 | 84% | 69/82 | 57% | 20/35 | 74% | 20/27 |

**Breakdown by Class**

| Name | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| Topping.java | 76% | 22/29 | 63% | 10/16 | 91% | 10/11 |

Figure 1: The original mutation score for the Topping class in Menu Microservice



**Pit Test Coverage Report**

**Package Summary**

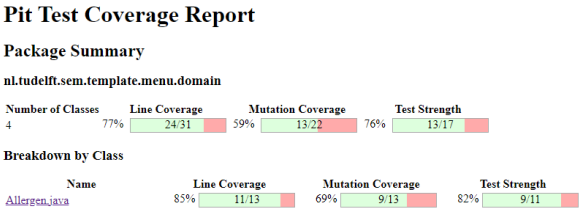
**nl.tudelft.sem.template.menu.domain.topping**

| Number of Classes | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| 4 | 93% | 76/82 | 69% | 24/35 | 80% | 24/30 |

**Breakdown by Class**

| Name | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| Topping.java | 100% | 29/29 | 88% | 14/16 | 100% | 14/14 |

Figure 2: The new mutation score for the Topping class in Menu Microservice



**Pit Test Coverage Report**

**Package Summary**

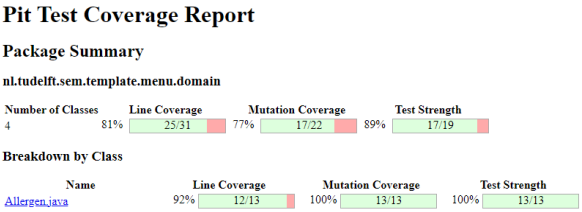
**nl.tudelft.sem.template.menu.domain**

| Number of Classes | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| 4 | 77% | 24/31 | 59% | 13/22 | 76% | 13/17 |

**Breakdown by Class**

| Name | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| Allergen.java | 85% | 11/13 | 69% | 9/13 | 82% | 9/11 |

Figure 3: The original mutation score for the Allergen class in Menu Microservice



**Pit Test Coverage Report**

**Package Summary**

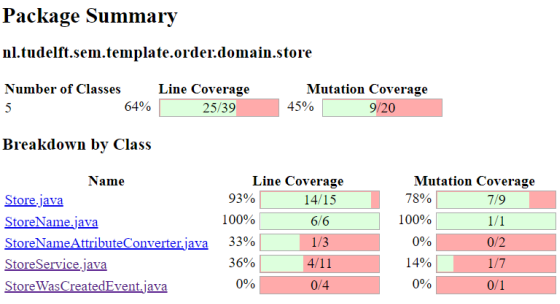
**nl.tudelft.sem.template.menu.domain**

| Number of Classes | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| 4 | 81% | 25/31 | 77% | 17/22 | 89% | 17/19 |

**Breakdown by Class**

| Name | Line Coverage | | Mutation Coverage | | Test Strength | |
|---|---|---|---|---|---|---|
| Allergen.java | 92% | 12/13 | 100% | 13/13 | 100% | 13/13 |

Figure 4: The new mutation score for the Allergen class in Menu Microservice



**Package Summary**

**nl.tudelft.sem.template.order.domain.store**

| Number of Classes | Line Coverage | | Mutation Coverage | |
|---|---|---|---|---|
| 5 | 64% | 25/39 | 45% | 9/20 |

**Breakdown by Class**

| Name | Line Coverage | | Mutation Coverage | |
|---|---|---|---|---|
| Store.java | 93% | 14/15 | 78% | 7/9 |
| StoreName.java | 100% | 6/6 | 100% | 1/1 |
| StoreNameAttributeConverter.java | 33% | 1/3 | 0% | 0/2 |
| StoreService.java | 36% | 4/11 | 14% | 1/7 |
| StoreWasCreatedEvent.java | 0% | 0/4 | 0% | 0/1 |

Figure 5: The original mutation score for the StoreService class in Order Microservice

| Number of Classes | Line Coverage | | Mutation Coverage | |
|---|---|---|---|---|
| 4 | 100% | 35/35 | 100% | 19/19 |

**Breakdown by Class**

| Name | Line Coverage | | Mutation Coverage | |
|---|---|---|---|---|
| Store.java | 100% | 15/15 | 100% | 9/9 |
| StoreName.java | 100% | 6/6 | 100% | 1/1 |
| StoreNameAttributeConverter.java | 100% | 3/3 | 100% | 2/2 |
| StoreService.java | 100% | 11/11 | 100% | 7/7 |

Figure 6: The new mutation score for the StoreService class in Order Microservice

## Breakdown by Class

| Name | Line Coverage | | Mutation Coverage | |
|---|---|---|---|---|
| AddToOrderService.java | 94% | 64/68 | 78% | 25/32 |
| BasicPricingStrategy.java | 100% | 9/9 | 100% | 6/6 |
| ModifyOrderService.java | 84% | 41/49 | 33% | 5/15 |
| Order.java | 91% | 29/32 | 75% | 6/8 |
| OrderData.java | 86% | 24/28 | 83% | 5/6 |
| OrderDataConverter.java | 4% | 1/23 | 0% | 0/13 |
| OrderHelperService.java | 100% | 28/28 | 75% | 9/12 |
| OrderRepository.java | 0% | 0/6 | 0% | 0/9 |
| OrderService.java | 74% | 29/39 | 50% | 8/16 |
| OrderStrategyConverter.java | 8% | 1/12 | 0% | 0/8 |
| OrderUtils.java | 88% | 35/40 | 57% | 17/30 |

Figure 7: The metrics for OrderUtils before adding and modding tests.

## Breakdown by Class

| Name | Line Coverage | | Mutation Coverage | |
|---|---|---|---|---|
| AddToOrderService.java | 94% | 64/68 | 78% | 25/32 |
| BasicPricingStrategy.java | 100% | 9/9 | 100% | 6/6 |
| ModifyOrderService.java | 84% | 41/49 | 33% | 5/15 |
| Order.java | 91% | 29/32 | 75% | 6/8 |
| OrderData.java | 86% | 24/28 | 100% | 6/6 |
| OrderDataConverter.java | 4% | 1/23 | 0% | 0/13 |
| OrderHelperService.java | 100% | 28/28 | 75% | 9/12 |
| OrderRepository.java | 0% | 0/6 | 0% | 0/9 |
| OrderService.java | 74% | 29/39 | 50% | 8/16 |
| OrderStrategyConverter.java | 8% | 1/12 | 0% | 0/8 |
| OrderUtils.java | 92% | 54/59 | 93% | 40/43 |

Figure 8: The metrics for OrderUtils after adding and modifying tests.

```
76      public Order getOrder(int orderId) throws OrderDoesNotExistException {
77          Optional<Order> order = orderRepository.findByOrderId(orderId);
78          if(order.isEmpty()) throw new OrderDoesNotExistException(orderId);
79          else return order.get();
80      }
```

Figure 9: The getOrder method before a mutation is introduced

```
public Order getOrder(int orderId) throws OrderDoesNotExistException {
    Optional<Order> order = orderRepository.findByOrderId(orderId);
    return order.get();
}
```

Figure 10: The getOrder method with the mutation being introduced

```
public void checkStatus(Order order) throws OrderIsAlreadyPaidException, OrderCancelledException {
    if(order.getData().getStatus().equals(Status.PAID)){
        throw new OrderIsAlreadyPaidException();
    }
    if(order.getData().getStatus().equals(Status.CANCELLED)){
        throw new OrderCancelledException(order.getOrderId());
    }
}
```

Figure 11: The checkStatus method before a mutation is introduced

```
97      public void checkStatus(Order order) throws OrderIsAlreadyPaidException, OrderCancelledException {
98  //      if(order.getData().getStatus().equals(Status.PAID)){
99  //          throw new OrderIsAlreadyPaidException();
100 //      }
101 //      if(order.getData().getStatus().equals(Status.CANCELLED)){
102 //          throw new OrderCancelledException(order.getOrderId());
103 //      }
104     }
```

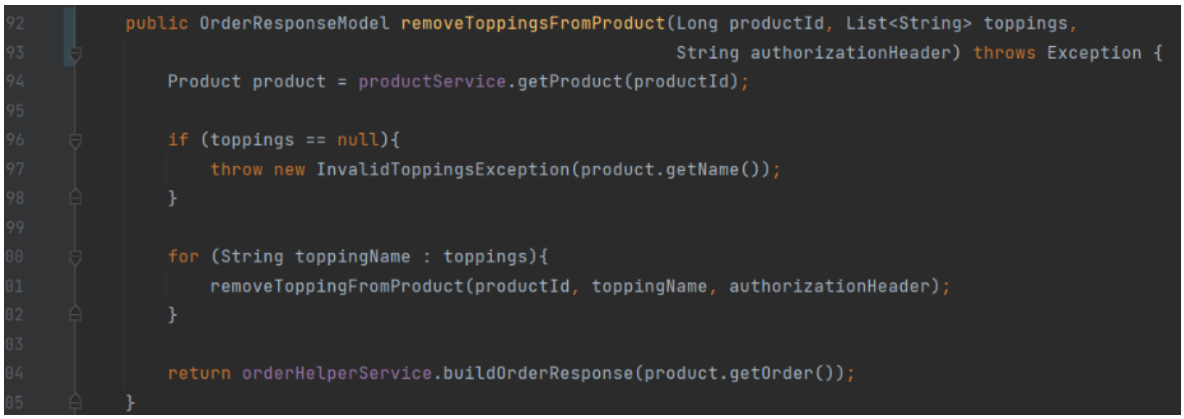
Figure 12: The checkStatus method with the mutation being introduced

```java
@PostMapping("/removeProductsFromOrder")
public ResponseEntity<List<OrderResponseModel>> removeProductsFromOrder(
        @RequestBody RemoveProductsFromOrderRequestModel request) {
    if (!authManager.getRole().equals("USER")) {
        throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Only Users can modify orders");
    }

    try {
        if (request.getProducts() != null) {
            for (Long productId : request.getProducts()) {
                Product product = productService.getProduct(productId);
                if (!orderService.orderBelongsToUser(product.getOrder().getOrderId(), authManager.getEmail())) {
                    throw new ResponseStatusException(HttpStatus.UNAUTHORIZED,
                            "This order was created by another account");
                }
            }
        }

        return ResponseEntity.ok().body(modifyOrderService.removeProductsFromOrder(request.getProducts()));
    } catch (ResponseStatusException e) {
        throw e;
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    }
}
```

Figure 13: The removeProductsFromOrder method before a mutation is introduced

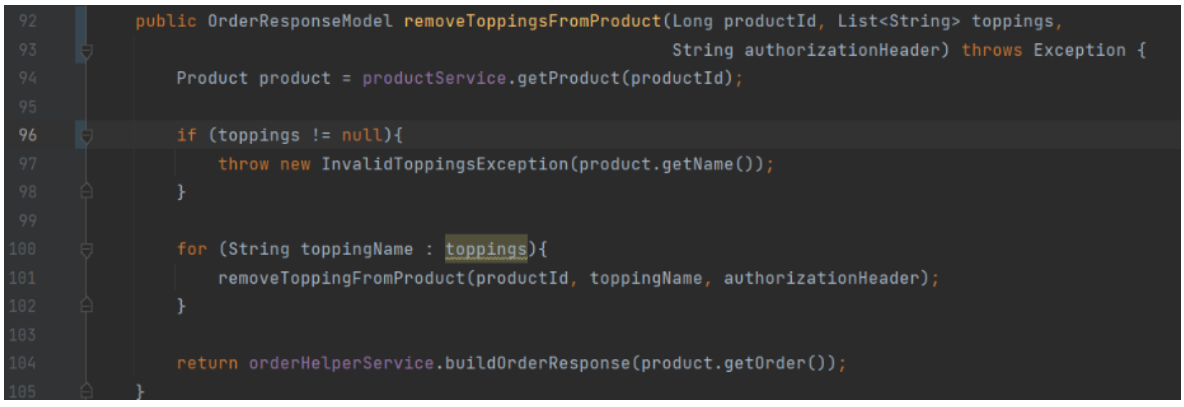
```java
@PostMapping("/removeProductsFromOrder")
public ResponseEntity<List<OrderResponseModel>> removeProductsFromOrder(
        @RequestBody RemoveProductsFromOrderRequestModel request) {
    if (!authManager.getRole().equals("USER")) {
        throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Only Users can modify orders");
    }

    try {
        if (request.getProducts() == null) {
            for (Long productId : request.getProducts()) {
                Product product = productService.getProduct(productId);
                if (!orderService.orderBelongsToUser(product.getOrder().getOrderId(), authManager.getEmail())) {
                    throw new ResponseStatusException(HttpStatus.UNAUTHORIZED,
                            "This order was created by another account");
                }
            }
        }

        return ResponseEntity.ok().body(modifyOrderService.removeProductsFromOrder(request.getProducts()));
    } catch (ResponseStatusException e) {
        throw e;
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
    }
}
```

Figure 14: The removeProductsFromOrder method with the mutation being introduced



```
92    public OrderResponseModel removeToppingsFromProduct(Long productId, List<String> toppings,
93                                                        String authorizationHeader) throws Exception {
94        Product product = productService.getProduct(productId);
95
96        if (toppings == null){
97            throw new InvalidToppingsException(product.getName());
98        }
99
00        for (String toppingName : toppings){
01            removeToppingFromProduct(productId, toppingName, authorizationHeader);
02        }
03
04        return orderHelperService.buildOrderResponse(product.getOrder());
05    }
```

Figure 15: The removeToppingsFromProduct method before a mutation is introduced



```
92    public OrderResponseModel removeToppingsFromProduct(Long productId, List<String> toppings,
93                                                        String authorizationHeader) throws Exception {
94        Product product = productService.getProduct(productId);
95
96        if (toppings != null){
97            throw new InvalidToppingsException(product.getName());
98        }
99
100       for (String toppingName : toppings){
101           removeToppingFromProduct(productId, toppingName, authorizationHeader);
102       }
103
104       return orderHelperService.buildOrderResponse(product.getOrder());
105   }
```

Figure 16: The removeToppingsFromProduct method with the mutation being introduced