Delft University of Technology

Software Engineering
Methods

Assignment 3: Mutation Testing

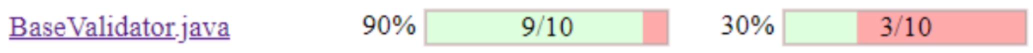
CSE2115	Group 31b
Alexandra Darie	5511100
Elena Dumitrescu	5527236
Xingyu Han	5343755
Zoya van Meel	5114470
Robert Vădăstreanu	5508096
Sander Vermeulen	5482127

January 25, 2023

Task 1: Automated Mutation Testing

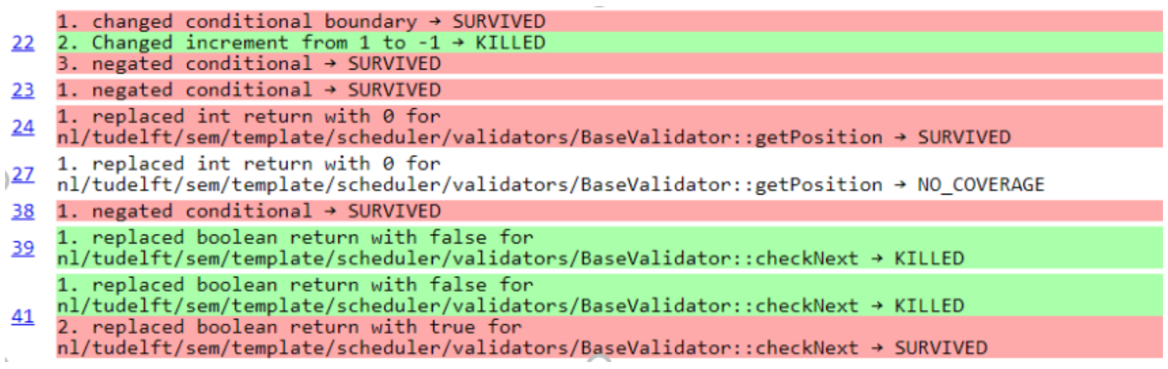
1.1 Validators - Scheduler microservice

Upon analyzing the Scheduler microservice's classes, we noticed that one of our Chain of Responsibility validators, the BaseValidator, had a pretty low mutation coverage (30%). Since these validators handle functionality crucial to our application, namely matching the users with appropriate events, we have decided to improve their test quality.



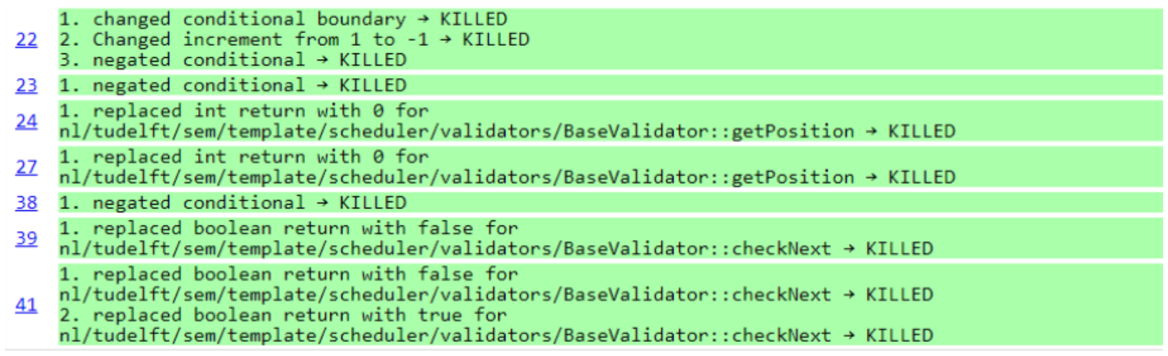
```
BaseValidator.java    90%  9/10    30%  3/10
```

The main issue with this class is that we had two improperly tested methods: getPosition() and checkNext(). They were indirectly checked in other tests, which explains why the line coverage is very high, but we did not have any test class dedicated to this validator.



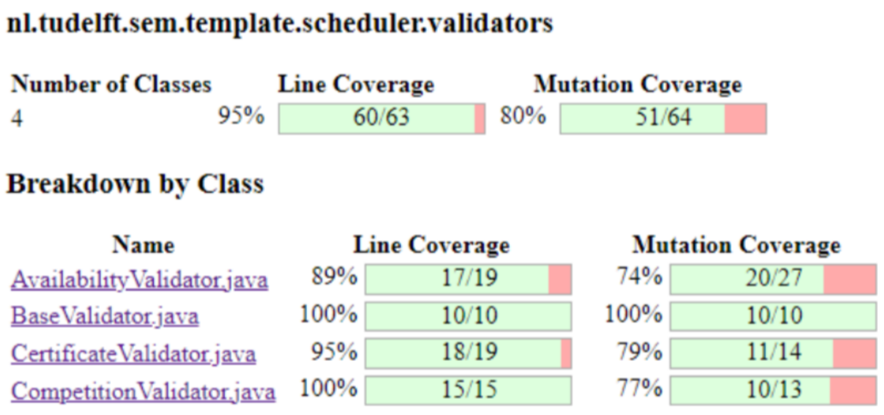
```
22  1. changed conditional boundary → SURVIVED
    2. Changed increment from 1 to -1 → KILLED
    3. negated conditional → SURVIVED
23  1. negated conditional → SURVIVED
24  1. replaced int return with 0 for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::getPosition → SURVIVED
27  1. replaced int return with 0 for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::getPosition → NO_COVERAGE
38  1. negated conditional → SURVIVED
39  1. replaced boolean return with false for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::checkNext → KILLED
41  1. replaced boolean return with false for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::checkNext → KILLED
    2. replaced boolean return with true for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::checkNext → SURVIVED
```

We have, therefore, added a BaseValidatorTest class, along with tests for both of the above methods. These tests ensure that the functionality performs correctly on basic input, as well as for edge cases. We used mocks to check the part of the functionality that moved to the next validator. As a result, the BaseValidator now has 100% mutation coverage. The main changes are contained in the commit #231186d9.



```
22  1. changed conditional boundary → KILLED
    2. Changed increment from 1 to -1 → KILLED
    3. negated conditional → KILLED
23  1. negated conditional → KILLED
24  1. replaced int return with 0 for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::getPosition → KILLED
27  1. replaced int return with 0 for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::getPosition → KILLED
38  1. negated conditional → KILLED
39  1. replaced boolean return with false for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::checkNext → KILLED
41  1. replaced boolean return with false for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::checkNext → KILLED
    2. replaced boolean return with true for
    nl/tudelft/sem/template/scheduler/validators/BaseValidator::checkNext → KILLED
```

The overall mutation and line coverage for all validators are now the following:



nl.tudelft.sem.template.scheduler.validators

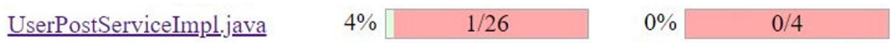
Number of Classes	Line Coverage		Mutation Coverage	
4	95%	60/63	80%	51/64

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
AvailabilityValidator.java	89%	17/19	74%	20/27
BaseValidator.java	100%	10/10	100%	10/10
CertificateValidator.java	95%	18/19	79%	11/14
CompetitionValidator.java	100%	15/15	77%	10/13

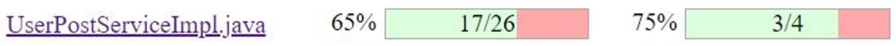
1.2 UserPostServiceImpl - User microservice

Using PITest, we noticed that both the mutation test coverage and line coverage of the UserPostServiceImpl class were 0% and 4% respectively, even though the class did have unit tests written for it. Upon further investigation, we found that the tests passed, but did not function properly. In theory, these tests would cover most mutants, but in practice they caught nothing. This is also reflected in the mutation score.



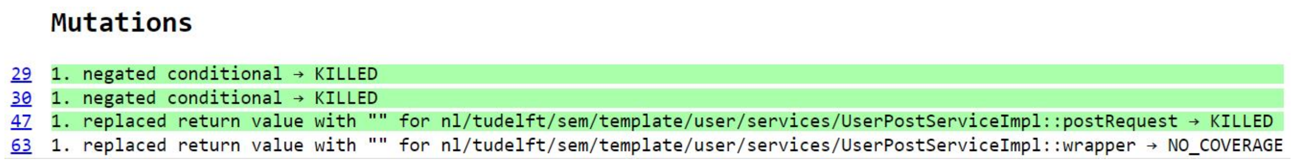
UserPostServiceImpl.java	4%	1/26	0%	0/4

The UserPostServiceImpl contains two methods: postRequest() and wrapper(). The wrapper() function was added to allow for mocking HTTP requests in unit tests and is never used on its own. The postRequest() function calls upon the wrapper function for the HTTP request to the scheduler microservice. After correcting the functionality of the unit tests, the mutation score rose to 75% and the line coverage to 65%. The corrected tests can be found inside commit #82d573f1. That commit introduced a failure in the pipeline which was then corrected in commit #38ef33fc. Applying mutation testing to this class got us to notice bugs in the test cases for the class instead of the class itself. These are bugs nonetheless and removing them gives us better assurance over the functionality of the class.



UserPostServiceImpl.java	65%	17/26	75%	3/4

Finally, there were four mutants generated by PITest, which can also be seen below. Three out of these mutants were handled by the test cases. The one mutant that survived, mutated the wrapper function. This mutant was left as-is because the wrapper function is needed for mocking HTTP requests.

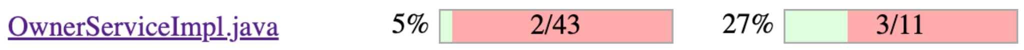


```
Mutations

29  1. negated conditional → KILLED
30  1. negated conditional → KILLED
47  1. replaced return value with "" for nl/tudelft/sem/template/user/services/UserPostServiceImpl::postRequest → KILLED
63  1. replaced return value with "" for nl/tudelft/sem/template/user/services/UserPostServiceImpl::wrapper → NO_COVERAGE
```

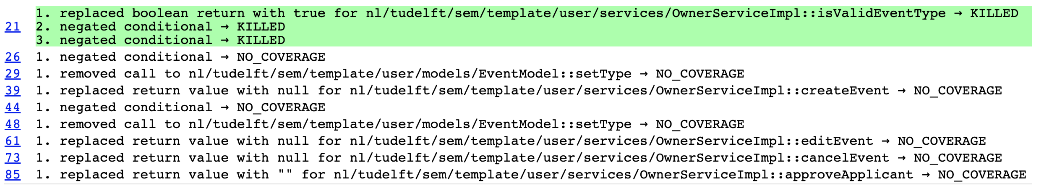
1.3 OwnerServiceImpl - User microservice

After running the PITest analysis tool on the OwnerServiceImpl class, which handles the event owner's functionality, we realised that it only has 27% mutation coverage. Having said that, we took a look over at the createEvent() and editEvent() methods and substracted some possible bugs w.r.t mutation testing. The PITest report before looked like this:



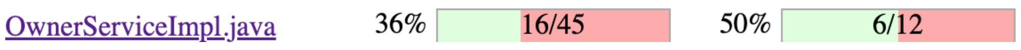
OwnerServiceImpl.java	5%	2/43	27%	3/11

The most important issue with this class was that the isValidEventType() method was checked outside of the other methods, and not checked for validation in the places where it is actually used. It was also lacking any checks for the requirements list in the EventModel object.

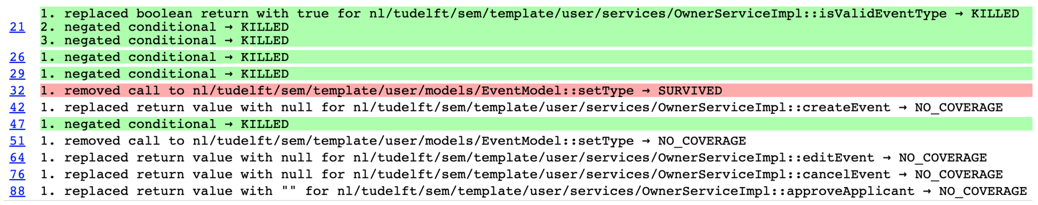


```
21  1. replaced boolean return with true for nl/tudelft/sem/template/user/services/OwnerServiceImpl::isValidEventType → KILLED
    2. negated conditional → KILLED
    3. negated conditional → KILLED
26  1. negated conditional → NO_COVERAGE
29  1. removed call to nl/tudelft/sem/template/user/models/EventModel::setType → NO_COVERAGE
39  1. replaced return value with null for nl/tudelft/sem/template/user/services/OwnerServiceImpl::createEvent → NO_COVERAGE
44  1. negated conditional → NO_COVERAGE
48  1. removed call to nl/tudelft/sem/template/user/models/EventModel::setType → NO_COVERAGE
61  1. replaced return value with null for nl/tudelft/sem/template/user/services/OwnerServiceImpl::editEvent → NO_COVERAGE
73  1. replaced return value with null for nl/tudelft/sem/template/user/services/OwnerServiceImpl::cancelEvent → NO_COVERAGE
85  1. replaced return value with "" for nl/tudelft/sem/template/user/services/OwnerServiceImpl::approveApplicant → NO_COVERAGE
```

After adding some tests for the createEvent() and editEvent() classes that can be seen in the commit #ca818d84, we can see that the mutation coverage was raised to 50%.



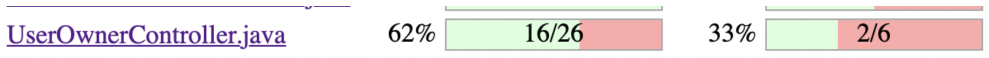
OwnerServiceImpl.java	36%	16/45	50%	6/12



```
21  1. replaced boolean return with true for nl/tudelft/sem/template/user/services/OwnerServiceImpl::isValidEventType → KILLED
    2. negated conditional → KILLED
    3. negated conditional → KILLED
26  1. negated conditional → KILLED
29  1. negated conditional → KILLED
32  1. removed call to nl/tudelft/sem/template/user/models/EventModel::setType → SURVIVED
42  1. replaced return value with null for nl/tudelft/sem/template/user/services/OwnerServiceImpl::createEvent → NO_COVERAGE
47  1. negated conditional → KILLED
51  1. removed call to nl/tudelft/sem/template/user/models/EventModel::setType → NO_COVERAGE
64  1. replaced return value with null for nl/tudelft/sem/template/user/services/OwnerServiceImpl::editEvent → NO_COVERAGE
76  1. replaced return value with null for nl/tudelft/sem/template/user/services/OwnerServiceImpl::cancelEvent → NO_COVERAGE
88  1. replaced return value with "" for nl/tudelft/sem/template/user/services/OwnerServiceImpl::approveApplicant → NO_COVERAGE
```

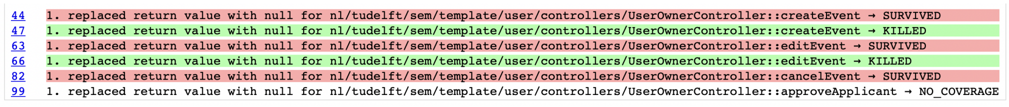
1.4 UserOwnerController - User microservice

After running the PITest analysis for the UserOwnerController class, which is the Controller for a user in the case that they are an owner, we realised that it only has 33% mutation coverage. This caused us to look further at the class and we found that the createEvent() and editEvent() methods had some possible bugs related to mutation testing. The PITest report before our improvements looked like this:



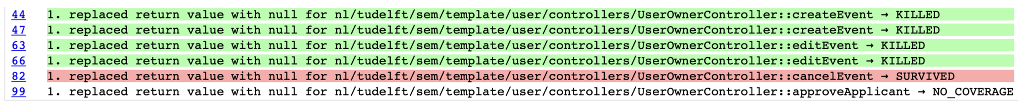
UserOwnerController.java	62%	16/26	33%	2/6

The mutants were related to a case when our methods return null. This was not properly tested and it is a situation that can appear when we do not receive any response to our requests.



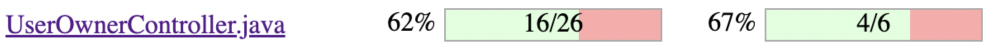
```
44  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::createEvent → SURVIVED
47  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::createEvent → KILLED
63  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::editEvent → SURVIVED
66  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::editEvent → KILLED
82  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::cancelEvent → SURVIVED
99  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::approveApplicant → NO_COVERAGE
```

After modifying some tests and checking for the special case discovered, which can be seen in commit #b5baac35, 2 of the mutants were killed:



```
44  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::createEvent → KILLED
47  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::createEvent → KILLED
63  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::editEvent → KILLED
66  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::editEvent → KILLED
82  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::cancelEvent → SURVIVED
99  1. replaced return value with null for nl/tudelft/sem/template/user/controllers/UserOwnerController::approveApplicant → NO_COVERAGE
```

The overall mutation coverage for the class, after our improvements, has been raised to 67% as also displayed below:



UserOwnerController.java	62%	16/26	67%	4/6

Task 2: Manual Mutation Testing

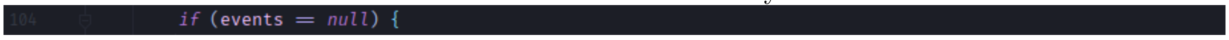
For the manual mutation testing, we have decided to choose the Scheduler domain, as the functionality of our whole application relies on its functionality. We have selected four different classes on which we have been able to find untested mutations:

2.1 EventServiceImpl

The EventServiceImpl class is an implementation class that builds the communication between the scheduler microservice and the event microservice. This class will hold the prerequisite processing of user-event matching which is the core function of our application.

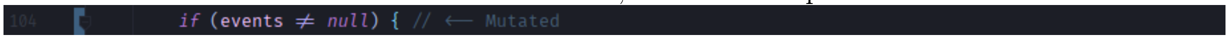
To do the manual mutation testing, a relational operator replacement is injected into the filterEvents() method, where we changed the '==' to '!='. This is also shown in the images below:

The intended functionality:



```
    if (events == null) {
```

The same line with the inserted mutation; the relation operator has been inverted:



```
    if (events ≠ null) { // ⟵ Mutated
```

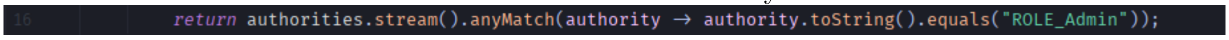
This relational statement decides whether to return an empty event list when there are no events provided by the event microservice, thus the statement plays as a critical assertion that validates the input and ensures the rest of logic can be executed as expected.

We have added tests which will now validate the correct behaviour, in the case that the given events are null. This can also be seen in commit #eb2e6361

2.2 AuthenticationHandler

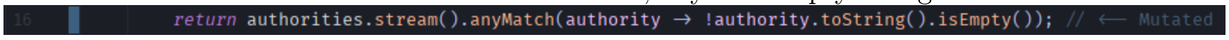
The AuthenicationHandler class handles the authentication in relation to determining if a user is an admin or not, so it is important this works as intended so we can ensure security for our system. We have found a mutation, in which case it would classify all users as admin. We were able to change the condition, from finding if the user has the admin role, to just checking if they have *any* role, which would still pass all of the tests. You can see the changed line with and without the injected mutant in the images below:

The intended functionality:



```
    return authorities.stream().anyMatch(authority → authority.toString().equals("ROLE_Admin"));
```

The same line with the inserted mutation; any non-empty string would match:



```
    return authorities.stream().anyMatch(authority → !authority.toString().isEmpty()); // ⟵ Mutated
```

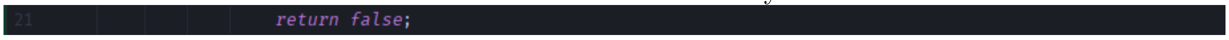
This mutation was not caught by any of our existing tests, so we made sure to test both the case in which a user is and is not an admin and thus being classified as one or the other. We have added these tests in commit #6a59c8a0.

2.3 CertificateValidator

The CertificateValidator class is a validator that controls whether a user has the required certificate to play a role in the coming event. According to the scenario description, a cox position for a user is only allowed when the user has earned a certificate for a boat type that is equal to or higher than the boat's required certificate. Otherwise, the position can not be filled. Thus, the validator controls our application's core business logic of user-position matching.

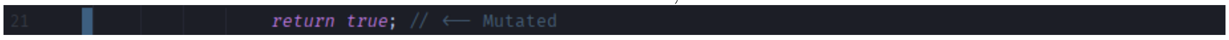
To do the manual mutation testing, a mutant is injected into the handle() method changing the return value of the second if statement from 'false' to 'true', as also shown in the images below:

The intended functionality:



```
            return false;
```

The same line with the inserted mutation; return value has been inverted:



```
            return true; // ⟵ Mutated
```

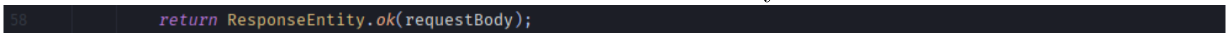
This if statement is a part of deciding whether the user has earned a valid certificate to get the cox position. After the mutation, the method will return true when the user does not have any certificate, which means the user can play as a cox without the certificate restriction and this is against the scenario.

To resolve this, we have added a test asserting that in the event that the user has no certificate, it will still return false. This can also be seen in commit #a9d2d088

2.4 SchedulerController

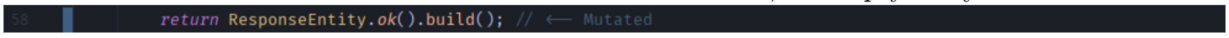
The SchedulerController class, and more specifically the '/match' endpoint that it implements, is crucial for our matching process and one of the most important functions of our application as a whole. Therefore, we have taken a closer look at this method and found that the tests did not cover all mutations we could introduce. More specifically: this endpoint is supposed to return the events which match the user's preferences, but we have found that if we introduce a mutation in which we do not actually return a body, it would still pass all tests. As the body of this endpoint contains crucial information needed in the matching process, it is important to ensure the body of this endpoint contains the right information. This mutation can be seen in the images below:

The intended functionality:



```
        return ResponseEntity.ok(requestBody);
```

The same line with the inserted mutation; an empty body:



```
        return ResponseEntity.ok().build(); // ⟵ Mutated
```

To fix this mutation, we have introduced an integration test which verifies that if the request is successfully executed, where it also asserts that the body of the response contains the right information. This can be seen in commit #ec7c4a29.