engineering proceedings

MDPI

Proceeding Paper

Using OpenSky Data for Teaching Software Engineering to Undergraduates †

Kai Renz

Department of Computer Science, Darmstadt University of Applied Sciences, Schöfferstr. 8b, 64295 Darmstadt, Germany; kai.renz@h-da.de

† Presented at the 10th OpenSky Symposium, Delft, The Netherlands, 10–11 November 2022.

Abstract: In the past year, the author has been using the OpenSky ressources for teaching purposes in the context of Software-Engineering courses for undergraduates at the Darmstadt University of Applied Sciences (Germany). Challenges in using the OpenSky-Online-API during hands-on sessions with groups of students could be found in the following areas: restrictions on the update frequency for data-retrieval calls when using the same IP address from the university network; necessity to create logins for accessing data for a specific timestamp; and a lack of student understanding of basic flight-data terminology, including geographic location. The OpenSky-Network data was used to teach Software-Engineering topics with the focus on professional software tests including UI-Testing with an OpenLayer visualization, testing of network connectivity using a circuit breaker, and the usage of mocks and stubs to obtain independent integration tests. To overcome the technological limitations of the data retrieval, a simple flight-data proxy was implemented to be used instead of the actual OpenSky website. This paper gives some insights into the specific testing topics covered by the course and the usage of OpenSky flight data. Some future improvements are suggested so that undergraduate students can further get to know the interesting field of flight-data processing while learning essential techniques for professional software testing.

Keywords: OpenSky; teaching computer science (software engineering); proxy for aircraft-position data; testing techniques; OpenLayers

Citation: Renz, K. Using OpenSky Data for Teaching Software Engineering to Undergraduate. *Eng. Proc.* **2022**, *28*, 3. https://doi.org/10.3390/engproc2022028003

Academic Editors: Michael Schultz and Junzi Sun

Published: 14 December 2022

1. Introduction

The need for a professional approach to the subject of testing software applications has now become part of all common software-engineering process models [1]. For many students, this subject area is uncharted territory at the beginning of their studies. To motivate the need for testing, it makes sense at a university of applied sciences to work with real-world data and a comprehensible task. The freely accessible data of the OpenSky network [2] represent an excellent opportunity to teach many aspects of software engineering in a lively fashion.

There exist already a number of excellent frameworks for handling OpenSky data, e.g., traffic [3] or OpenSky-traffic-viz [4]. However, the focus of the course is less on the deeper analysis of flight data and more on the aspects of data retrieval via standard interfaces such as REST and testing techniques for the implemented software. Therefore, these frameworks are not used in this particular setting.

While working with the OpenSky data via the API at https://opensky-network.org/api/states/all (accessed on 13 December 2022) during hands-on sessions with groups of up to 16 students, the following challenges where discovered:

- Restrictions on the update frequency for data-retrieval calls when using the same IP address from the university network;
- The necessity to register for an OpenSky account in order to retrieve data for a specific timestamp;

- A lack of student understanding of basic flight-data terminology, including geographic location.

The first two challenges could easily have been resolved by asking for a special treatment for the university access using a registered IP address or other means of identification. However, the author decided to use a more sustainable approach: because it is not necessary to use real-time data to learn software-engineering techniques, previously recorded flight data was provided via a locally hosted flight-data proxy.

The last challenge was not so easy to solve. Understanding and dealing with flight data requires a large amount of knowledge, which admittedly can only be built up by occupying oneself with the matter over a longer period of time. The approach to teaching the material to the students was to link common techniques of software testing to the actual processing of flight data.

During the 13-week course, the following testing techniques were among those taught (students spent an average of 4 h per week on the tasks):

- Dependency injection and test doubles (including mocks and stubs);
- Interface testing using REST assured;
- Behaviour-driven development;
- Test automation for a sample flight-track web application using TestCafe [5] (including headless tests);
- Using a continuous integration pipeline which includes automated tests.

Figure 1 shows an overview of the software components that were used for the course. The testing components are drawn with a red background. Almost all of the software (including testing) was run in docker containers, which created additional difficulties for some students but at the same time reduced the complexity of setting up a common infrastructure. Additionally, all tests where included in a continuous integration pipeline.

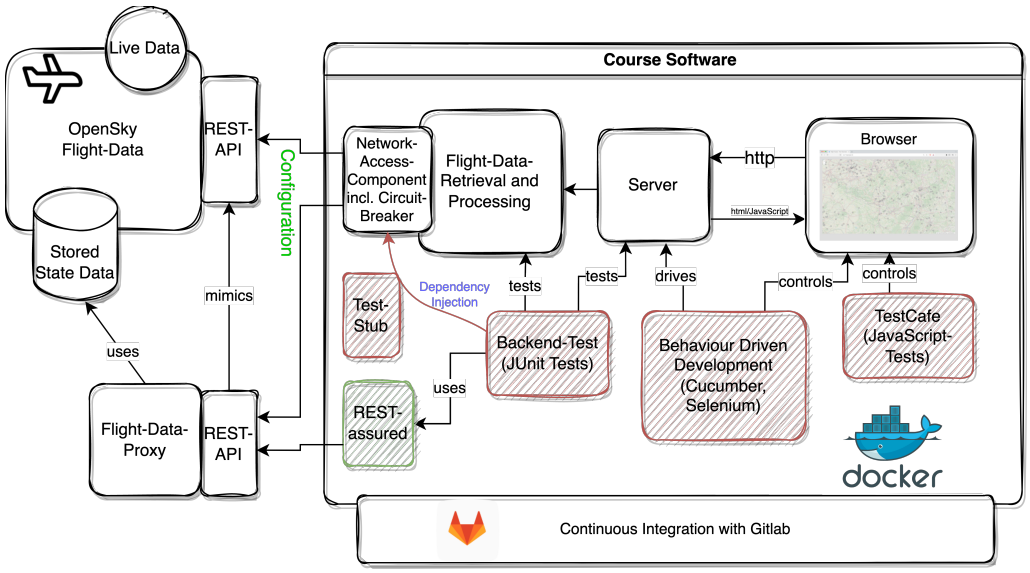

Figure 1. Overview of components and technologies.

2. Flight-Data Proxy

In order to be independent of any limitations while accessing the OpenSky data, a simple flight-data proxy was implemented. Using one hour of recorded data (which is available through https://opensky-network.org/datasets/, accessed on 13 December 2022) the proxy determines which flight data to return to a caller by matching the current minute and second to the equivalent time in the hour of the pre-recorded data. This means that the same hour of air-traffic is repeated every hour. This only reveals itself during the transition from the last minute of an hour to the first minute of the following hour. This limitation has not led to any problems during the course.

The flight-data proxy currently mimics the behaviour of calls to https://opensky-network.org/api/states/all (accessed on 13 December 2022) without any additional parameters (the equivalent call to the flight-data proxy is https://flugwege.de/flightdata, accessed on 13 December 2022).

The flight-data proxy is not only used for testing the API (see the following sections) but also used as a data provider for a simple web application which is also used for testing purposes (see Section 6).

As can be seen in Figure 1, the network-access component can be configured to use either the original OpenSky-REST-API or the REST-API of the flight-data proxy.

3. Dependency Injection and Mocking

Dependency Injection is a well-known design pattern for creating loosely coupled software which is maintainable and testable [6]. Course participants were introduced to this concept by being given the task of implementing a data-retrieval module that needed to access the flight-data proxy via the REST-API. In order to be able to test the flight-data processing without the need for a working network connection and also to have control over the test data, the students needed to use a dependency injection to inject a test double (this case required the use of a stub [1] (pp. 93)) for provisioning the flight data. The test stub in Figure 1 implements the same interface as the network-access component and is injected into the flight-data retrieval component by the backend test.

The flight-data stub had to provide the same data that a call to the API would retrieve over the network. This helped the students understand what data needed to be provided and also helped them figure out what the flight-data processing unit needed to do.

An additional technique that was taught during this exercise was the use of test-driven development (TDD). Although most students have heard of this technique, there is often some resistance to actually using it. The authors' experience shows that the use of mocks and stubs encourages the application of TDD and helps students to appreciate its value for professional software testing.

4. Testing REST Interfaces with REST-Assured

In order for the students to learn how to test REST-full APIs, the well-established tool REST-assured (https://github.com/rest-assured/rest-assured, accessed on 13 December 2022) was used in the course. During the implementation of the tests, the following issues needed to addressed by students:

- What kind of request is required for the desired functionality (GET, POST, PUT, or DELETE)?
- What is the expected answer?
- What happens if the network connection is not working? Are there defined values for network timeouts?
- Are values retrieved over the network checked for errors? Are there any vulnerabilities that might be exploited if the server has been compromised?

A small sample of a REST-assured call to check the provided data can be seen in Figure 2

```
given.filter(sessionFilter).get(path: "http://localhost:13471/flights").peek()
        .then().body(is("{\"time\":1523627930,\"flights\":[" +
        "{\"latitude\":52.1423,\"longitude\":9.45133," +
        "\"height\":731.234,\"icao24\":\"123456\",\"callsign\":\"DLH543\","+
        "\"heading\":20.3,\"lastUpdate\":1523627920}]}"));
```

Figure 2. Sample code for a REST-assured based test.

5. Behaviour-Driven Development

Behaviour-driven development (BDD) is an approach to testing using a language for tests which non-developers can understand [7]. A commonly used implementation for this

scheme is cucumber using the testing language called 'gherkin' (https://cucumber.io/, accessed on 13 December 2022). Tests are placed in 'feature' files which are then processed by cucumber and transformed into test-function calls. A sample for such a test can be seen in Figure 3.

```
1  Feature: Connect
2
3  Scenario: This is the first cucumber sencario
4
5  Given There is no connection to opensky-network.org
6  When I start the application
7  Then the returned flightData is equal to "{}"
```

Figure 3. Simple test written in gherkin.

In the context of the course, BDD was used for testing features in the backend and the frontend. The latter was implemented using the end-to-end-testing tool Selenium (https://www.selenium.dev/, accessed on 13 December 2022). This approach helped the students to understand and implement the required test infrastructure, where a test server and a test browser had to be activated in order for the tests to function. An additional challenge was the integration of the cucumber tests in the continuous integration pipeline.

6. Test Automation for a Flight-Track Web Application

A very important part of the course is the teaching of how to setup end-to-end tests using test automation tools for web browsers. There are many tools freely available: Cypress, Selenium, TestCafe, and others. For this course, TestCafe (https://testcafe.io/, accessed on 13 December 2022) was chosen, because it was easy to setup and the tests can be understood and written quite easily.

A very simple flight-tracking web application was implemented using the OpenLayer-Framework (Version 7.1, https://openlayers.org/ (accessed on 13 December 2022) in conjunction with the flight-data proxy. This application is shown in Figure 4 and can be accessed at https://flugwege.de (accessed on 13 December 2022). The flight positions are updated every 10 s using a websocket connection.

A pretty difficult task for the students was to implement a complete testing stage, which included a dummy flight-track server and a browser which was running in headless mode being controlled by TestCafe.

Many students also had trouble finding an intentionally placed bug in the web application: The rotation of the flight symbols was just carried out using the *true_track* value of the state-vector directly. The OpenLayer-API expects graphic symbols (icons) to be rotated using a radian value. When pointing out this issue, the importance of reading and understanding the used data and APIs was immediately understood by the students.

Other issues with regards to web-application testing where only briefly addressed:

- Asserting the validity of the drawn map and the contained flight symbols (including location and rotation);
- Asserting the update of flight-positions at the expected rate.

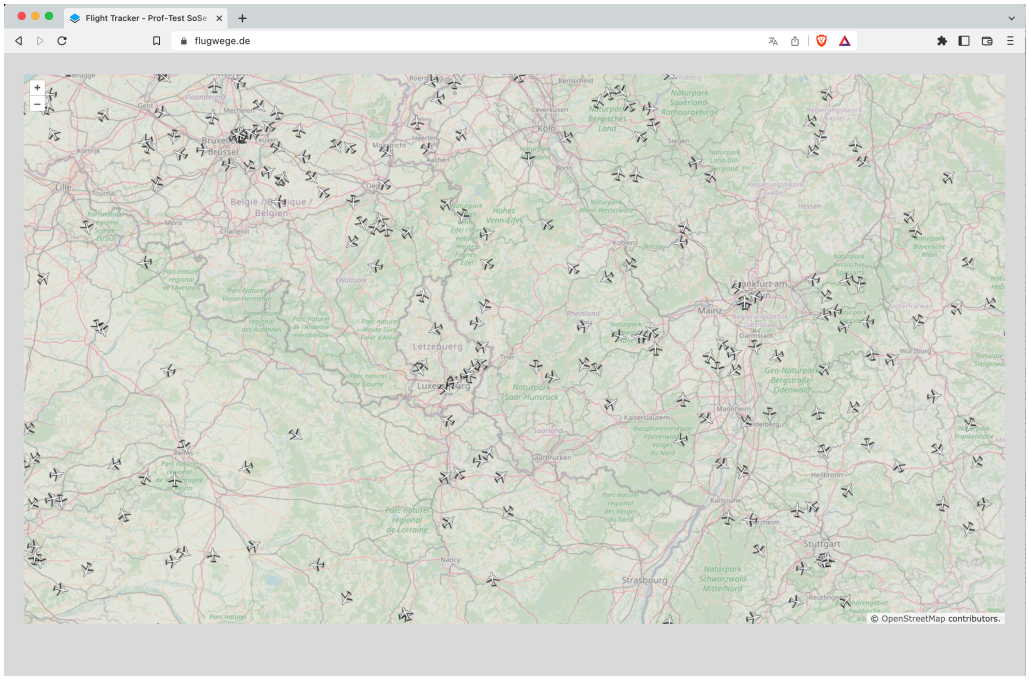

Figure 4. Flight-tracking web application.

7. Results

The evaluation of the course by the students was consistently very positive. A summary of the evaluation can be seen in Figure 5: the number range on the right-hand side goes from 1 (very positive) to 6 (negative). An English translation of the different areas was added. Significant feedback revealed that the practical reference to the flight data was very helpful and resulted in learning the relevant techniques well.

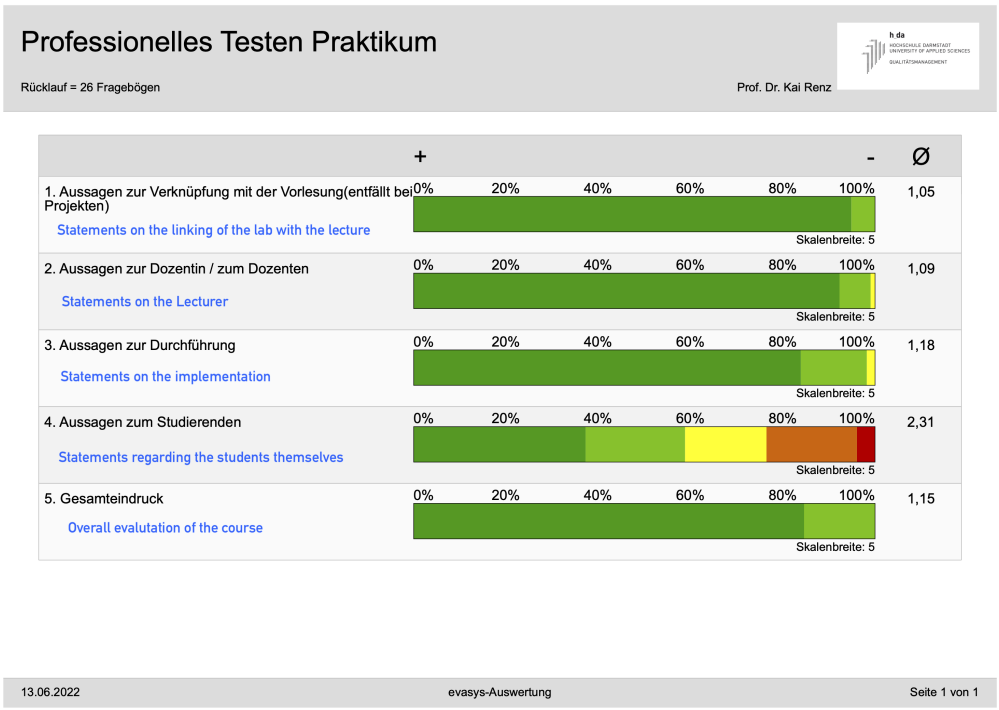

Figure 5. Summary of student evaluation of the course.

Some students noted that the subject matter (especially the handling of flight data) was quite demanding. Unfortunately, due to the heavy workload of students in other courses,

additional engagement with the topic was not observed. The bug described in Section 6 concerning the incorrect rotation of the aircraft symbols was not fixed by all students, although the display then remained incorrect.

Overall, the number of test methods and tools used in the course is quite high. This poses a certain challenge for the students. However, it is also possible to learn many different approaches and then apply them in other contexts.

Some aspects of the course not mentioned in this article also deal with techniques for testing asynchronous code or testing multi-threading.

8. Conclusions and Further Work

For the future of the course, it is planned to focus the use of the test tools even more on the underlying software architecture. Currently, the tasks are partially detached from each other and it is sometimes not so easy for the students to classify the topics in relation to each other.

In addition, the author would like to incorporate the aspect of end-to-end testing more deeply. In particular, the issues of the correct representation of flight-data symbols on the website should be implemented in the future with newer approaches (e.g., image recognition and AI).

Due to time constraints, the area of mobile testing is also currently not addressed in more detail on the course. Especially for this area, the flight data of the OpenSky network is particularly suitable and should be considered in the future.

Furthermore, it is planned to make the course materials available as open source in order to establish an exchange with other interested teachers (and students).

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Khorikov, V. *Unit Testing Principles, Practices, and Patterns*; Manning: Birmingham, UK, 2020.
2. Schäfer, M.; Strohmeier, M.; Lenders, V.; Martinovic, I.; Wilhelm, M. Bringing up OpenSky: A large-scale ADS-B sensor network for research. In Proceedings of the 13th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2014), Berlin, Germany, 15–17 April 2014.
3. Olive, X. Traffic, a toolbox for processing and analysing air traffic data. *J. Open Source Softw.* **2019**, *4*, 1518. [CrossRef]
4. Dubot, T. Python Scripts to Load and Visualize OpenSky Network Air Traffic Data. 2022. Available online: https://github.com/thomasdubdub/opensky-traffic-viz (accessed on 3 November 2022).
5. Shpakovskyi, D. *Modern Web Testing with TestCafe-Get to Grips with End-To-End Web Testing with TestCafe and JavaScript*; Packt Publishing Ltd: Birmingham, UK, 2020.
6. Seemann, M.; Deursen, S.V. *Dependency Injection Principles, Practices, and Patterns*; Simon and Schuster: New York, NY, USA, 2019.
7. Wynne, M.; Hellesoy, A.; Tooke, S. *The Cucumber Book-Behaviour-Driven Development for Testers and Developers*; Pragmatic Bookshelf: Raleigh, NC, USA; Dallas, TX, USA, 2017.