Informatics 1: Object Oriented Programming

Tutorial 03

Week 5: Software Testing 10/02 - 14/02

Volker Seeker (`volker.seeker@ed.ac.uk`)
Vidminas Mikucionis (`vmikucio@ed.ac.uk`)

1   Introduction

Today's topic is software testing. We have already seen and done lots of software testing ourselves by now (for example, in the previous tutorial, checking whether Sudoku World works after changing something also counts!)

While the worst case for a cute little game like Sudoku World is crashing, real life software defects could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and even recent history is full of such examples:

- Nissan had to recall over 34 million cars from the market due to software failure in the airbag sensory detectors. They have been linked to 11 deaths in the USA, as of January, 2018.

  http://www.nissanproblems.com/takata/

- In December 2014, some of the Amazons third party retailers saw their product prices were reduced to 1p due to a software glitch. Some small businesses lost up to 100,000 as a result.

  https://www.edwincoe.com/blogs/main/amazon-sellers-seek-recovery-of-financial-losses-after-glitch-sees-items-sold-for-1p/

- In April 2015, Bloomberg terminal in London crashed due to a software glitch. This potentially affected more than 300,000 traders on financial markets and forced the UK government to postpone a 3bn debt sale.

  https://www.theguardian.com/business/2015/apr/17/uk-halts-bond-sale-bloomberg-terminals-crash-worldwide

- On 24 April, 2015, Starbucks' suffered a massive outage that shut down the point-of-sales systems at stores across the U.S. and Canada. Because transactions were not working, stores gave away tea and coffee to customers for free, which cost the company an estimated $3 to $4 million

  https://www.geekwire.com/2015/starbucks-lost-millions-in-sales-because-of-a-system-refresh-computer-problem/

2   So what can we do about it?

Writing software that can be 100% verifiably correct is infeasible in most[1] cases. Programs are just too big to check every single possible input and action!

---

[1]You can learn more about this by looking up "Formal Verification"

But not all is lost - by testing the most important functionality, you can still catch lots of bugs and ensure that this functionality works, even if it does not cover everything. Testing can only prove if bugs exist - not their absence!

We will start by trying black box testing. Black box testing is a technique which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

Task 1 - Time to get our hands dirty

Figure 1: Let's try it ourselves[2]

An anonymous customer has requested a black box review of their mysterious program called Digitiser. They sent in this specification: "This is a program that should get an integer from user input and print out the digits in that integer, one by one."

> ✎ Download the **BlackBoxTest.jar** from the Inf1B Learn page: https://course.inf.ed.ac.uk/inf1b

Once you have downloaded BlackBoxTest.jar, run it from the command line with `java -jar BlackBoxTest.jar`.

You have about 5 minutes to try various inputs and see if you can break the program!
Hint: there are multiple ways to make it crash and also for some inputs, the program does not work correctly

Now, it's time to fix the issues you have found! The tutors will share a link to the Java source file of this program, so that you can edit it.

> ✎ Download the **Digitiser.java** from the Inf1B Learn page: https://course.inf.ed.ac.uk/inf1b

> ✎ Optional trivia: there is a technique, called reverse engineering, which allows you to reconstruct source code from a given executable. With Java, this is particularly easy to do - you can open up a JAR file with an archive manager as it is just a regular zip archive. There, you can find compiled .class files, which you can extract and open in IntelliJ IDEA, which will de-compile them back into Java source code.

Task 2 - White Box and Unit testing

White box testing is another software testing technique. Can you guess what it is?

It's the opposite of black box testing in a way - instead of testing program functionality without looking at the code, it is all about checking the internal program logic. Does a method always return the correct output for specific inputs? Can it handle a bad input or can it be broken in some way?

Unit testing is one way of approaching white box testing. It works by designing tests that check a single (hence unit) functionality of the program.

---

[2]Image source: https://www.anniebabymonitor.com/mydata/myuploads/sites/4/2016/11/foto37.jpg

For example, if we had a program that returns the first element of an array, we could write some unit tests that pass in different arrays, e.g. [1, 2, 3] or [3, 2, 1] or [0, 0] and check that the returned result is 1, 3 and 0 respectively. It would also be wise to check what happens if you pass in an array with just a single element, e.g. [5] or an empty array.

Figure 2: Not this kind of unit![3]

☞ Note, that when a test fails, that does not necessarily mean the functionality is wrong - sometimes the test itself can be wrong too!

✎ Download the **BinaryToDecimal.zip** from the Inf1B Learn page: https://course.inf.ed.ac.uk/inf1b

Open the BinaryConverter project in an IDE. You will find there is a main class BinaryConverter.java which contains all the functionality, as well as a skeleton unit test framework BinaryConverterTest.java.

Your task is to fill in the unit tests (see the comments in the skeleton file for guidance). Run them and note all issues - then fix the problems in BinaryConverter.java until all tests pass.

Figure 3: Not this kind of white box![4]

The main benefit of unit testing is that once the tests are written, running them can be automated. Rather than having to check all these inputs by hand, you can run all the checks with a tool and have it report back to you some statistics - how many tests are passing, how many are failing and which ones.

If you test your program just once - this isn't much of an improvement, setting up the unit testing framework takes some time after all - why not just use print statements? However, in a larger project, when you change a piece of functionality, it will often affect many other modules. It is a good idea to run tests every time a change is made to ensure that nothing new was broken (this is called a regression). With unit tests, this is as simple as clicking a button.

3 Air quality in the UK

It is time to see software testing in action with a real-life project!

With growing awareness about climate change, more and more data becomes available about various aspects of pollution. We are going to look at information published by the UK Government Department for Environment, Food and Rural Affairs (Defra) Air Information Resources (AIR) (you can find their website here https://uk-air.defra.gov.uk/).

Our task will be making a program that can determine the air quality in Edinburgh on particular day based on provided data.

---

[3]Image source: https://i.ytimg.com/vi/s-ciCFAaBwM/maxresdefault.jpg
[4]Image source: https://image.shutterstock.com/image-photo/image-450w-665616367.jpg

Monitoring site	Running 8 Hour mean Ozone (μgm^{-3})	Hourly mean Nitrogen dioxide (μgm^{-3})	Max 15 min mean Sulphur dioxide (μgm^{-3})	Running 24 Hour mean PM$_{2.5}$ Particles (μgm^{-3})	Running 24 Hour mean PM$_{10}$ Particles (μgm^{-3})	Last updated
Ealing Horn Lane Timeseries Graph	n/m	n/m	n/m	n/m	24 (2 Low) *1	11/11/2019 11:00
Eastbourne Timeseries Graph	n/m	n/a	n/m	13 (2 Low)	18 (2 Low) *3	11/11/2019 11:00
Edinburgh Nicolson Street Timeseries Graph	n/m	54 (1 Low)	n/m	n/m	n/m	11/11/2019 11:00
Edinburgh St Leonards Timeseries Graph	50 (2 Low)	3 (1 Low)	9 (1 Low)	4 (1 Low)	6 (1 Low) *3	11/11/2019 11:00

Figure 4: Edinburgh air quality data at 11am on 11/11/19

3.1 Let's find Edinburgh!

Automatic monitoring stations measure air contents at regular intervals throughout the UK.

At the time of writing this tutorial, there were 2 such stations in Edinburgh. Let's find them ourselves!

First visit https://uk-air.defra.gov.uk/networks/find-sites.

In step 1, you will be asked to select the monitoring network. The Automatic Urban and Rural Monitoring Network (AURN) is the one we want.

For step 2, type in the first half of an Edinburgh post code (you can use your own post code if you live in Edinburgh, or just pick the university - its post code is EH8 9AJ, so you just want EH8). Then select the search area to be 100 square kilometres. The interactive map should update to show a square around the region you selected.

Finally, click "search network". This will bring you to a page with some information about the monitoring sites in Edinburgh. Perhaps you have been walking past these every day without even noticing they were there? The more you know...

What do these sites measure? You can find their latest sensor information on this page https://uk-air.

`defra.gov.uk/latest/currentlevels`, just click "View by monitoring site" and look for the sites you found earlier.

It should look something like the screenshot in figure 4.

3.2 The Daily Air Quality Index

While having data from sensors available is great, seeing values, such as $54 \mu g m^{-3}$ of Nitrogen Dioxide isn't very informative or helpful for most people.

To make this data easier to understand, the Daily Air Quality Index (DAQI) is used in the UK (see figure 5 for a descriptive overview).

The overall air pollution index for a site or region is determined by the highest concentration of five pollutants: Nitrogen Dioxide, Sulphur Dioxide, Ozone, Particles $< 2.5 \mu m$ ($PM_{2.5}$) and Particles $< 10 \mu m$ (PM_{10}).

Table 6 shows the details of how each pollutant contributes to the DAQI index.

For example, the Edinburgh St Leonards Street sensor on 11/11/2019 at 11:00 reported that there were $50 \mu g m^-3$ of ozone in the air. Using the table you can find that this falls in the $34 - 66 \mu g m^-3$ band, which is index 2 (and that's exactly what we see in the parentheses in figure 4). All the other pollutant figures map to index 1. Edinburgh is not too bad! :)

Overall, the DAQI for Edinburgh St Leonards Street on 11/11/2019 at 11:00 would be 2, because this is the highest index out of the 5 polluntants.

Air Pollution Banding	Value	Accompanying health messages for at-risk individuals*	Accompanying health messages for the general population
Low	1-3	**Enjoy** your usual outdoor activities.	**Enjoy** your usual outdoor activities.
Moderate	4-6	Adults and children with lung problems, and adults with heart problems, **who experience symptoms**, should **consider reducing** strenuous physical activity, particularly outdoors.	**Enjoy** your usual outdoor activities.
High	7-9	Adults and children with lung problems, and adults with heart problems, should **reduce** strenuous physical exertion, particularly outdoors, and particularly if they experience symptoms. People with asthma may find they need to use their reliever inhaler more often. Older people should also **reduce** physical exertion.	Anyone experiencing discomfort such as sore eyes, cough or sore throat should **consider reducing** activity, particularly outdoors.
Very High	10	Adults and children with lung problems, adults with heart problems, and older people, should **avoid** strenuous physical activity. People with asthma may find they need to use their reliever inhaler more often.	**Reduce** physical exertion, particularly outdoors, especially if you experience symptoms such as cough or sore throat.

Figure 5: You can find this at https://uk-air.defra.gov.uk/air-pollution/daqi

Table 1: The revised Daily Air Quality Index.

Band	Index	Ozone Running 8 hourly mean μgm^{-3}	Nitrogen Dioxide hourly mean μgm^{-3}	Sulphur Dioxide 15 minute mean μgm^{-3}	PM$_{2.5}$ Particles (EU Reference Equivalent) 24 hour mean μgm^{-3}	PM$_{10}$ Particles (EU Reference Equivalent) 24 hour mean μgm^{-3}
Low	1	0-33	0-67	0-88	0-11	0-16
Low	2	34-66	68-134	89-177	12-23	17-33
Low	3	67-100	135-200	178-266	24-35	34-50
Moderate	4	101-120	201-267	267-354	36-41	51-58
Moderate	5	121-140	268-334	355-443	42-47	59-66
Moderate	6	141-160	335-400	444-532	48-53	67-75
High	7	161-187	401-467	533-710	54-58	76-83
High	8	188-213	468-534	711-887	59-64	84-91
High	9	214-240	535-600	888-1064	65-70	92-100
Very High	10	241 or more	601 or more	1065 or more	71 or more	101 or more

Figure 6: Table from "Report: Update on Implementation of the Daily Air Quality Index" (https://uk-air.defra.gov.uk/library/reports?report_id=750)

Task 1 - Time to get started on our program!

For this exercise, get into pairs, as it is meant for pair programming. You can divide up the navigator and driver roles as you like. You will have about 10 minutes to investigate the project, and 15 minutes to test it and fix any issues.

Finally, the remainder of the tutorial will be all yours to implement additional features as you see fit.

Figure 7: The people demand change[5]

Suppose the UK government is in a crisis - the people demand climate action and DEFRA (Department for Environment, Food and Rural Affairs) are not managing to keep up.

You have been taken onboard to help DEFRA automate some of their work, so that they don't need to map air quality sensor data to DAQI levels by hand.

Your predecessor, a former developer, has started work on this project, but she mysteriously disappeared before it was complete. Luckily, the department was able to salvage the project source code from her hard drive. You can download the provided project framework from here:

> ✎ Download the **air-quality.zip** project from the Inf1B Learn page: https://course.inf.ed.ac.uk/inf1b

On the developer's desk, there were some handwritten notes, including a todo list. It said:

- Investigate failing JUnit tests
- Fill in missing functions (see TODO notes in source files)
- Test plotting sample data
- Try more recent data
- Add more tests
- What else?

Task 2 - Optional: Try it with your own data set

There was also another document with instructions for obtaining air quality sensor data:

1. Visit https://uk-air.defra.gov.uk/data/data_selector
2. Pick search hourly networks
3. Pick measured data for data type and hit save
4. Pick a date range (at first try something small, e.g. just last week)
5. Select monitoring sites by local authority, find City of Edinburgh and select all, then save selection
6. Pick pollutant by name and select all
7. For output type, choose data to email address (CSV), type in your email address and accept the terms and conditions (after reading them through, of course!)

---

[5]Image source: http://www.cellonline.org/wp-content/uploads/2016/02/can-transparent-500-png24.png

8. Click "Get Data" on the right-hand side menu

At the end of it, a handwritten scribbled note says "You will receive two files in your email - AirQuality-DataHourly.csv and AirQualityData15Minutes.csv. You can try opening both (with Excel or similar), but most likely the 15 minute one will mostly have "no data" entries, so the hourly document is much more useful. Rename it, copy it to the project test resources directory and change the data source filename in RealDataTest.java to use it".

Measurements from the sensors are published daily, including the assigned DAQI band, so you can compare your results with what can be found here https://uk-air.defra.gov.uk/data/DAQI-regional-data (although this website only allows seeing the DAQI for all of Central Scotland combined only, which may be slightly different to just Edinburgh - it should not be wildly different however, at most 1 DAQI level apart).

Task 3 - Advanced optional: Different data sources

If you have managed to figure everything out, you can try using another data source from: http://www.scottishairquality.scot/data/data-selector. This website has data from more sensors, at the time of writing there were Edinburgh Currie, Edinburgh Glasgow Road, Edinburgh Gorgie Road, Edinburgh Nicolson Street, Edinburgh Queensferry Road, Edinburgh Salamander St, Edinburgh St John's Road, Edinburgh St Leonards, Edinburgh Tower Street.

Since this data is formatted in a different way, this will require adjusting how the data is parsed by editing the CSVHelper class and is not part of the course, but if you are feeling particularly adventurous, feel free to try it out.

4 Further learning

In future courses, you can choose to learn data science techniques for analysing data like this (and others) in more interesting ways, make forecast predictions for the future, identify unusual patterns and classify the data into different categories.

This is what powers maps forecast maps like the one at: https://uk-air.defra.gov.uk/
If you enjoyed this topic, definitely check out the courses about data science and machine learning.

Also, if you liked investigating air pollution, you can learn more on the EU Commission website https://ec.europa.eu/environment/air/cleaner_air/
There is an interactive map of air pollution data for most of Europe available here: http://airindex.eea.europa.eu/
A similar map for the US can be found here: https://www.epa.gov/outdoor-air-quality-data/interactive-map-air-quality-monitors
You can study more about the environment and specifically from a computer science perspective by taking courses on Geographical Information Systems (GIS).

Figure 8: Thumbs up for Data Science[6]

---

[6]Image source: https://upload.wikimedia.org/wikipedia/commons/thumb/7/74/Beijing_bouddhist_monk_2009_IMG_1486.JPG/800px-Beijing_bouddhist_monk_2009_IMG_1486.JPG